Proyecto Fin de Carrera
Grado en Ingeniería de Tecnologías Industriales

Programación mediante herramientas visuales de microprocesadores digitales de señal (DSP) para aplicaciones de convertidores electrónicos de potencia.

Autor: Guillermo Hidalgo González
Tutor: Sergio Vázquez Pérez

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Carrera
Ingeniería de Tecnologías Industriales

Programación mediante herramientas visuales de microprocesadores digitales de señal (DSP) para aplicaciones de convertidores electrónicos de potencia.

Autor:
Guillermo Hidalgo González

Tutor:
Sergio Vázquez Pérez
Profesor titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: Programación mediante herramientas visuales de microprocesadores digitales de señal (DSP) para aplicaciones de convertidores electrónicos de potencia.

Autor: Guillermo Hidalgo González

Tutor: Sergio Vázquez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mis padres Lola y Guillermo, por sus ánimos y su apoyo incondicional durante estos años. Y a Paula, por todas las mañanas y tardes de biblioteca compartidas.

Agradecimientos

Para comenzar, me gustaría agradecer a mi tutor Sergio Vázquez Pérez, por proponerme un proyecto en el que trabajar, del cuál tras muchas horas de esfuerzo ha finalizado en este Trabajo fin de Grado.

Mencionar el buen ambiente de trabajo del departamento de Electrónica, donde Eduardo, Abraham y otros compañeros me han provisto de los útiles y componentes necesarios, así como de los conocimientos y consejos que me han ayudado para finalizar este proyecto.

A mi familia por su apoyo incondicional, y ánimos cuando más lo necesitaba.

Gracias.

Guillermo Hidalgo González

Sevilla, 2019

Resumen

El convertidor elevador de tensión, o dispositivo Boost, es un elemento esencial en la electrónica de potencia, el cuál a un nivel de tensión dado en la entrada, proporcionará en los bornes de salida una tensión mayor. El elevador actúa como enlace entre las plantas de generación de eléctrica renovable, teniendo más peso en las instalaciones fotovoltaicas cuya implementación está en aumento actualmente. Para llevar a cabo su funcionamiento requieren de un sistema de control para que puedan operar correctamente, y sin peligro de dañar el equipo.

Por esto, se programará un control en bucle cerrado a través de un microcontrolador mediante la herramienta simulink proporcionada por Matlab, el cual se encargará de generar la señal de modulación de los transistores del circuito a través de una serie de medidas tomadas. Estas medidas serán: la tensión de entrada, la tensión de salida y la intensidad que circula por la bobina del elevador, y se recogerán a través de los puertos ADC del microcontrolador.

El diagrama de simulink proporciona una visualización más esquemática del programa instalado en el dispositivo, además de permitir un grado alto de interacción, a través de unos actuadores digitales, con los que se podrá poner en marcha el circuito entre los distintos estados de funcionamiento.

No obstante, antes de mostrar los resultados obtenidos por el circuito real, se va a realizar una simulación con un circuito virtual con lo que posteriormente se compararán los resultados obtenidos.

Se construirá físicamente el dispositivo elevador con elementos eléctricos y electrónicos para operar el sistema en desarrollo, comprobando así que el sistema es operativo.

La coordinación de la toma de datos del circuito y su visualización en el ordenador, así como el envío de órdenes y comandos del mismo a la DSP se realizará gracias a una interfaz gráfica diseñada por la herramienta GUIDE de Matlab.

Abstract

This proyect has been written, with the purpose of desingning a Boost controller.

The control works through an interaction, between a Texas instrument microcontroller (model TMS320 F28377S) wich have the program installed with Matlab Simulink diagram, and a command window, created with GUIDE tool from Matlab.

The boost circuit will be built with electrical and electronical elements, and check for the correct operation of themselves with GUIDE views which consist in two axis windows and the numeric representation of the values.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

Vd	Tensión de entrada del circuito.
Vo	Tensión de salida del circuito.
IL	Intensidad por la bobina del circuito.
D	Duty cycle.
ADC	Analogical to Digital Converter.
DC/DC	Convertidor de señal Continua-Continua.
PWM	Pulse-Width Modulation, Modulación por ancho de pulsos.

1 INTRODUCCIÓN

En la actualidad, la búsqueda de una energía limpia con la que abastecer el sistema eléctrico del mundo se ha vuelto una prioridad. En las instalaciones fotovoltaicas el primer elemento de electrónica de potencia aparece justo a la salida del parque solar, es el Convertidor DC/DC Elevador.

1.1. Microcontrolador TMS320-F28377S

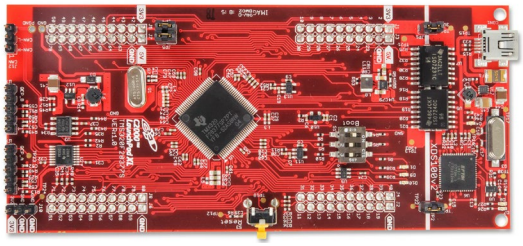


Figura 1-1. Microcontrolador TMS320-f28377S.

El objetivo de este Proyecto de Fin de Grado es la puesta en funcionamiento de un Elevador de Tensión a través de dicho microcontrolador, haciendo uso de la herramienta Simulink, además del desarrollo de una interfaz con la que se podrán visualizar las medidas tomadas del circuito real, así como enviar las órdenes o instrucciones necesarias para su correcto funcionamiento.

El microcontrolador F28377S proporcionará los elementos necesarios para el control que será implementado. Dispone de un total de 12 módulos PWM, cada uno con dos salidas A, B, con los que se envían las señales de modulación a los transistores del Elevador.

Para llevar a cabo este control es necesario tomar 3 medidas eléctricas del circuito: la tensión de entrada V_d, la tensión de salida V_o y la intensidad de la bobina I_L, que se llevará a cabo usando 3 módulos ADC de los 15 que dispone el microcontrolador.

1.2. Convertidor Elevador construido.

Para el montaje del convertidor elevador, se dispondrá de una serie de circuitos de adaptación de señal, así como un circuito puente en H, fuentes de tensión y elementos eléctricos como una bobina y cables. Las medidas de ADC se tomarán gracias a lectores de tensión e intensidad.

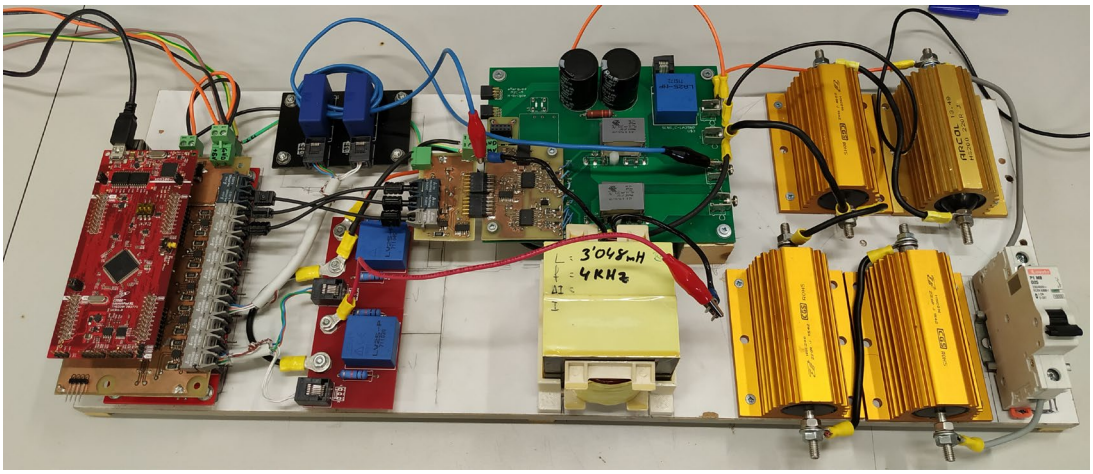


Figura 1-2. Convertidor boost montado.

2 CONVERTIDOR ELEVADOR DE TENSIÓN

Un Elevador de Tensión o dispositivo Boost es un convertidor tipo DC/DC que, a una tensión dada en sus bornes de entrada, genera una mayor en los bornes de salida. Está compuesto por una inductancia L, dos interruptores IGBT (con diodo) y una reactancia C.

Su modo de operación se basa en la modulación de los transistores, los cuales se controlan con una señal tipo Duty Cycle, operando de forma complementaria (mientras que uno está cerrado, el otro se encuentra abierto).

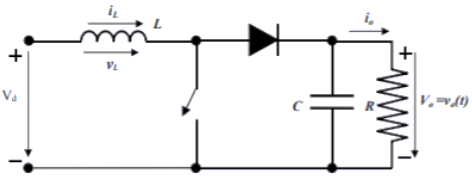


Figura 2-1. Diagrama convertidor Elevador.

El convertidor Elevador de tensión puede trabajar en dos estados de funcionamiento:

Modo de conducción Continua.

Definimos el Duty Cycle, como el tiempo que está operando el primer interruptor, de manera que podemos despejar la siguiente ecuación.

$$\begin{cases} V_d * t_{on} + (V_d - V_o) * t_{off} = 0 \\ T_s = t_{on} + t_{off} \end{cases}$$

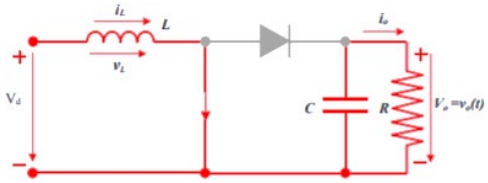
$$V_d * D + (V_d - V_o) * (1 - D) = 0$$

$$\frac{V_o}{V_d} = \frac{T_s}{t_{off}} = \frac{1}{1 - D}$$

El Convertidor trabaja en dos estados:

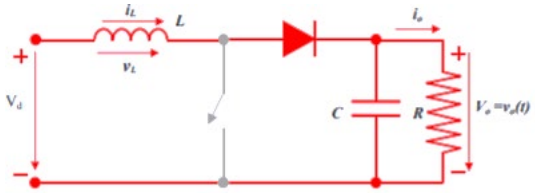
Intervalo 1. Conducción.

$$0 < t < Ts * D \ (t_{on})$$



Intervalo 2. No conducción.

$$t_{on} < t < T_s$$



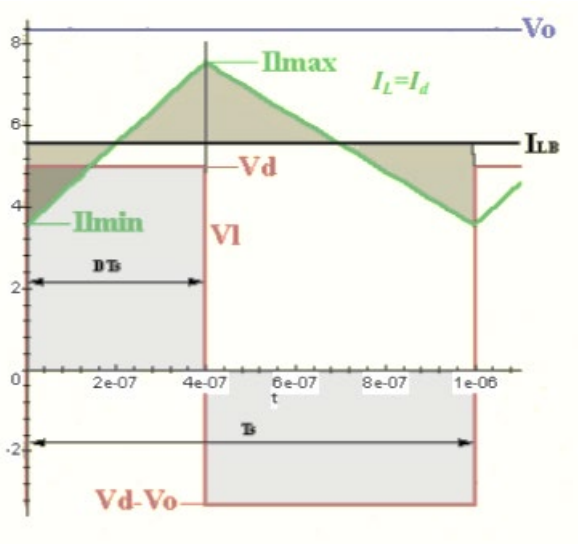


Figura 2-2. Estados de operación del convertidor Elevador (1). Mediciones del circuito en Continua (2).

Donde finalmente la Tensión a la salida del convertidor será:

$$V_o = V_d * \frac{1}{1-D}$$

Modo de conducción Discontinua.

Este modo de funcionamiento se alcanza cuando el convertidor llegar al límite entre los modos de conducción.

Este límite se alcanza cuando el valor de la intensidad de la bobina vale la mitad de la intensidad máxima que circula por ella, esto es:

$$I_{LB} = \frac{1}{2} i_{lmax} = \frac{T_S * V_o}{2 * L} * D * (1-D) = 4 * D * (1-D) * I_{LBmáx}$$

En el modo de conducción discontinua la intensidad de la bobina adquiere un valor nulo en cada periodo de funcionamiento. Este sería negativo, pero la circuitería lo impide.

El valor del Duty Cycle se complica, calculándose ahora:

$$\frac{V_o}{V_d} = \frac{\Delta_1 + D}{\Delta_1}$$

Siendo la Intensidad calculada del triángulo de la Figura 2-3 (2):

$$I_d = \frac{V_d}{2 * L} * (D + \Delta_1) * T_S * i_{lmáx}$$

Si no hay pérdidas se obtiene:

$$\frac{I_o}{I_d} = \frac{\Delta_1 + D}{\Delta_1}$$

(1)

Se calcula al valor del Duty Cycle:

$$D = \left(\frac{4}{27} * \frac{V_o}{V_d} * \left(\frac{V_o}{V_d} - 1\right) * \frac{I_o}{I_{oBmáx}}\right)^{1/2}$$

(2)

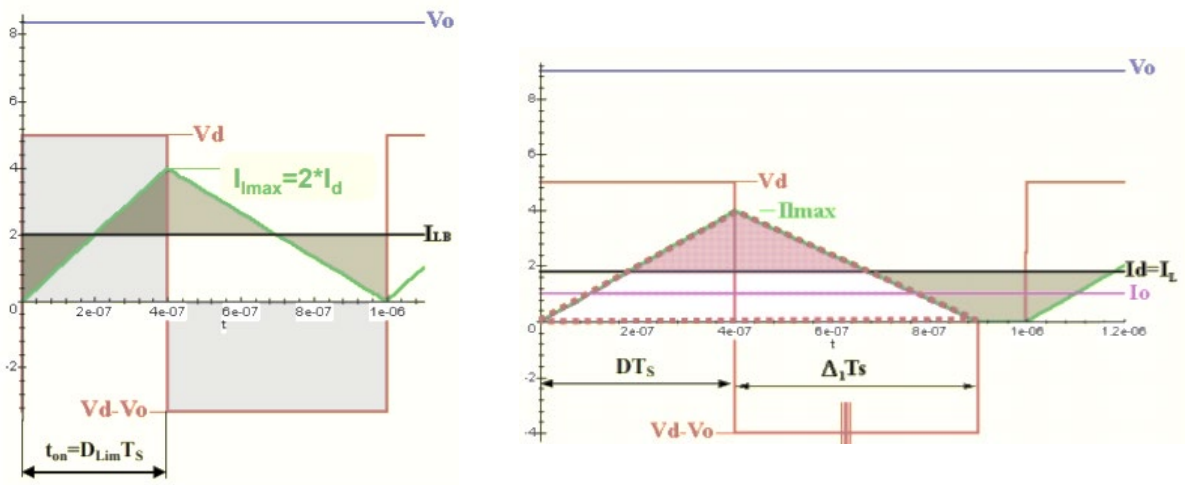


Figura 2-3. Mediciones del circuito en límite de conducción (1). Mediciones del circuito en Discontinua (2).

3 PASOS PREVIOS. PROGRAMAS

3.1. Programas necesarios.

El microcontrolador va a ser programado mediante la herramienta Simulink de Matlab, la cual permitirá una programación por bloques, donde es posible observar gráficamente las funciones que se programan.

Por esto, es necesario instalar una serie de programas y extensiones en el ordenador de trabajo.

1. Matlab, versión 2017a.
2. Code Composer Studio, versión v8.1.
3. Control Suite.
4. C2000 ware.
5. Toolchain: Code Generation Tools v6.4.6.
6. DSP2804x_v130.exe.

Una vez estos programas se encuetren instalados, se ejecuta Matlab, en la pestaña de Add-Ons y abrimos la opción Get Hardware Support Packages.

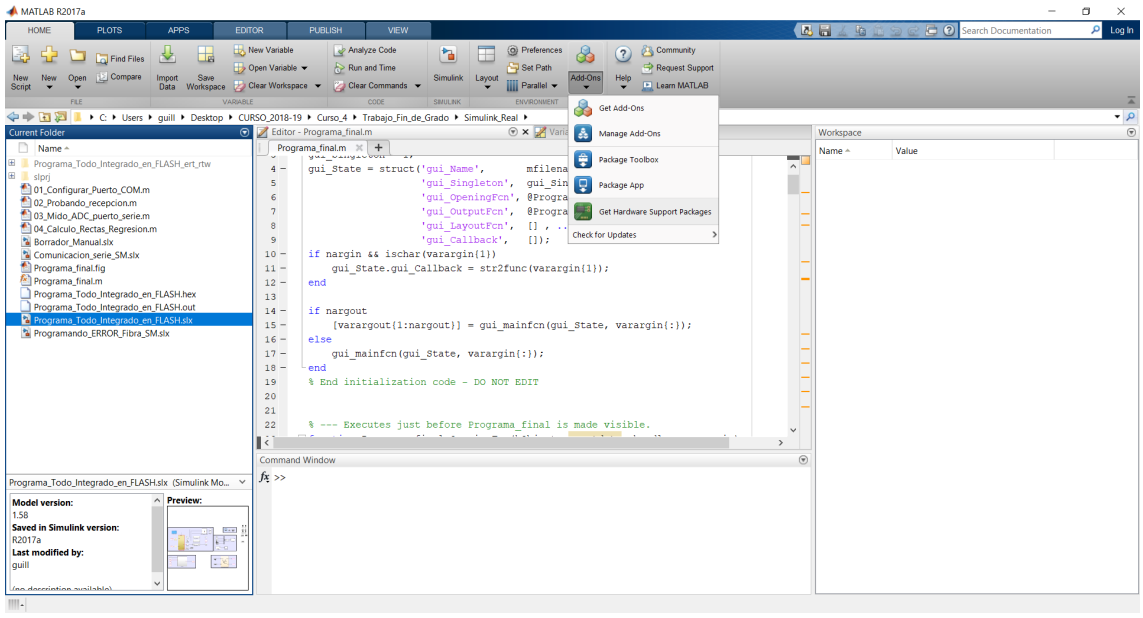


Figura 3-1. Guia instalación “Embedder Coder Support for TI C2000”. Get Hardware Support Packages.

Abierto el navegador, es necesario buscar “Embedder Coder Support for TI C2000”, e iniciar el instalador.

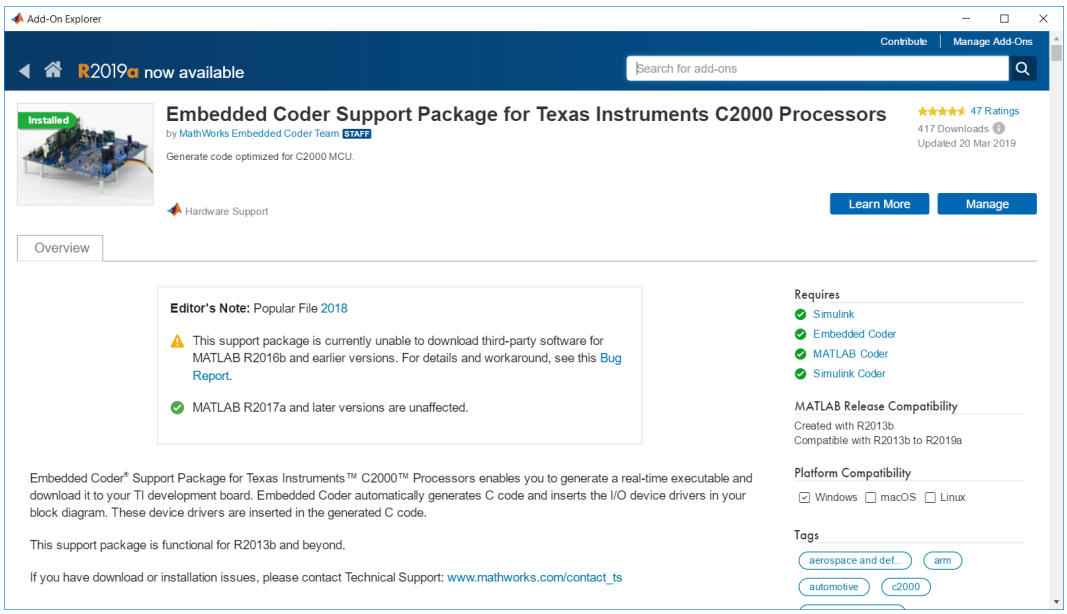


Figura 3-2. Guia instalación “Embedder Coder Support for TI C2000”. Extensión Matlab.

Hay que enlazar los programas y archivos previamente instalados de Texas Instrument, así como CCS.

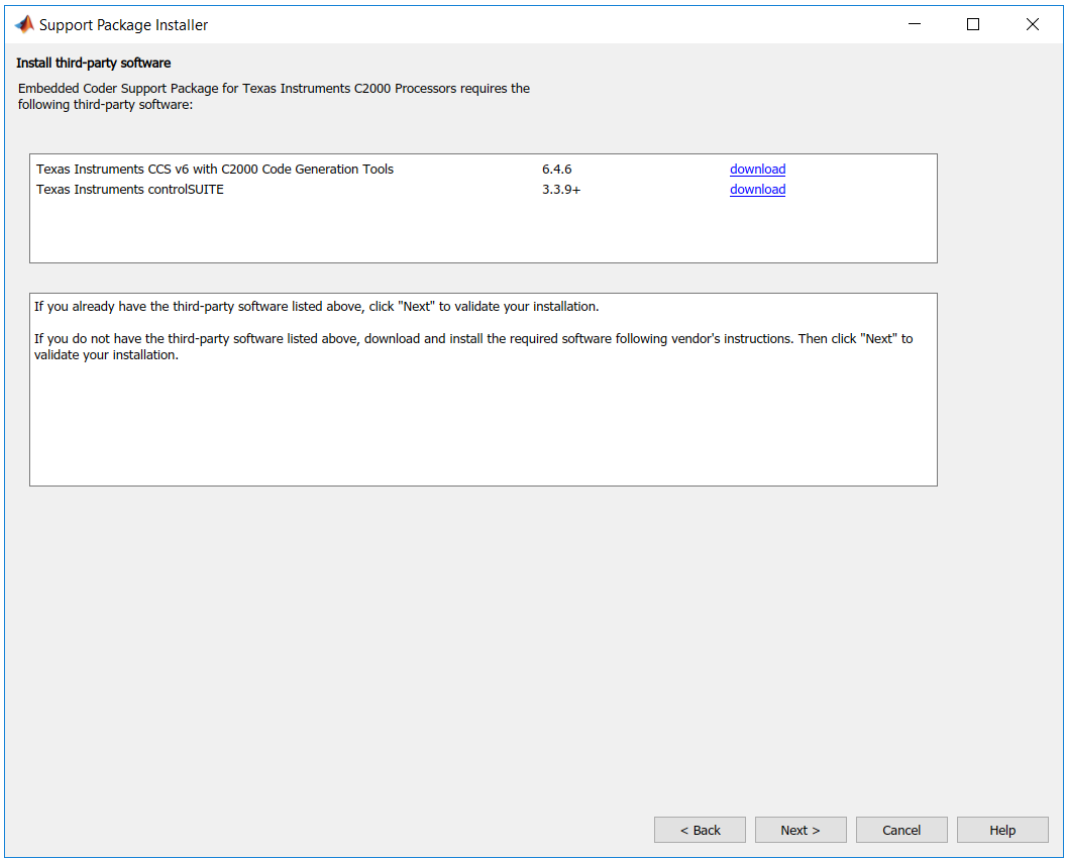


Figura 3-3. Guia instalación “Embedder Coder Support for TI C2000”. Procesos Instalación.

Una vez instalado, se podrá acceder a una serie de documentos y ejemplos con los que tomar contacto y probarlos en el microcontrolador.

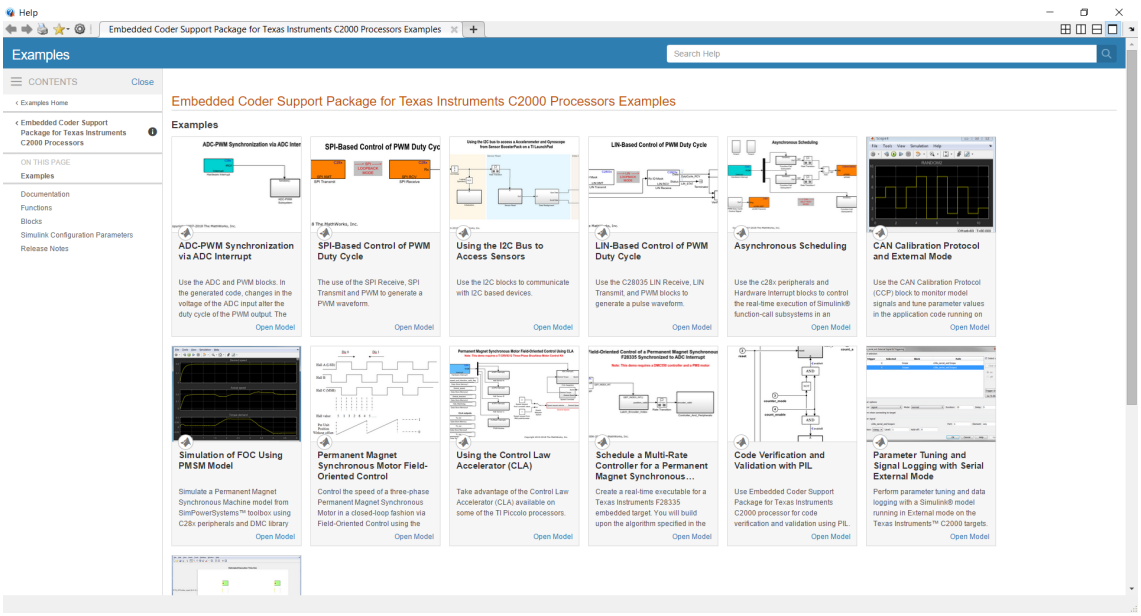


Figura 3-4. Guia instalación “Embedder Coder Support for TI C2000”. Ejemplos de Matlab.

3.2. Comunicación serie con el ordenador.

Para configurar la comunicación entre el microcontrolador y el ordenador, se siguen los siguientes pasos.

- Se ejecuta el administrador de dispositivos.
    - Puertos COM y LPT.
        - XDS100 Class USB Serial Port (COMX).

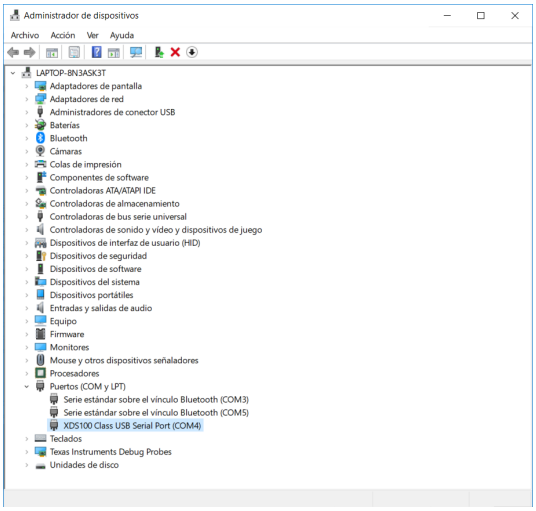


Figura 3-5. Puerto COM asociado al microcontrolador.

El puerto COMX es el puerto que más adelante hay que seleccionar en la configuración del modelo de simulink. Click derecho, accediendo a propiedades. En configuraión de puerto:

- Bits por segudo: 115200.
- Bits de datos: 8.
- Paridad: Ninguno.
- Bits de parada: 1.
- Cotrol de flujo: Ninguno.

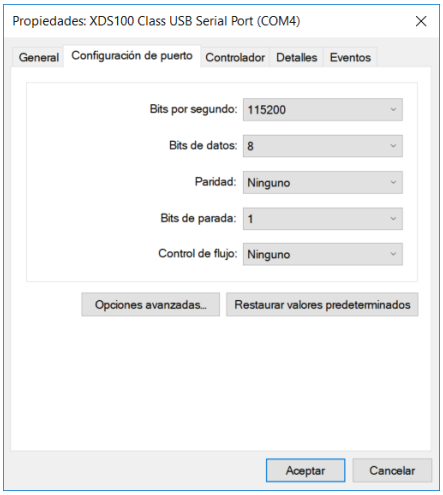


Figura 3-6. Configuración del puerto COMx asociado al microcontrolador.

3.3. Configuración del modelo en Matlab.

Completos los apartados anteriores e instalados los programas en el ordenador de trabajo, se puede abrir la herramienta Simulink y comenzar a trabajar.

Se crea un proyecto de Simulink y antes de agregar ningún bloque, hay que especificar en los ajustes del modelo una serie de características.

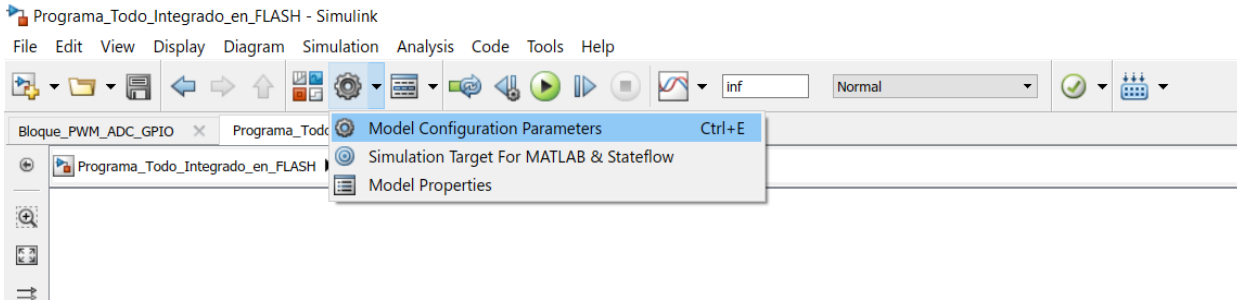


Figura 3-7. Configuración del proyecto de Simulink. Model Configuration Parameters.

- Model Configuration Parameters.
    - Solver
        - Solver options.
            - Type: Fixed-step.
            - Solver: discrete (no continuous states).
            - Fixed-step size (fundamental sample time): auto.

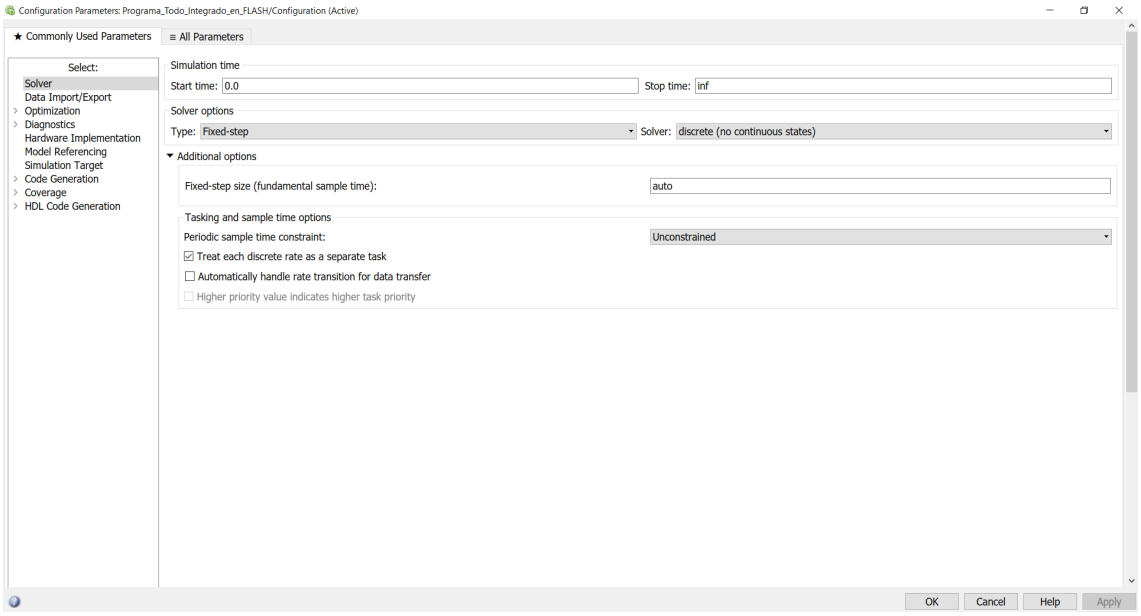


Figura 3-8. Configuración del proyecto de Simulink. Solver.

- Hardware implementation
  - Hardware board: TI Delfino F2837xS
  - Target Hardware Resources
    - Build Options.
      - Device Name: F28377S.
      - Activar: Boot from flash (stand-alone execution).
      - Activar: Use custom linker comand file.

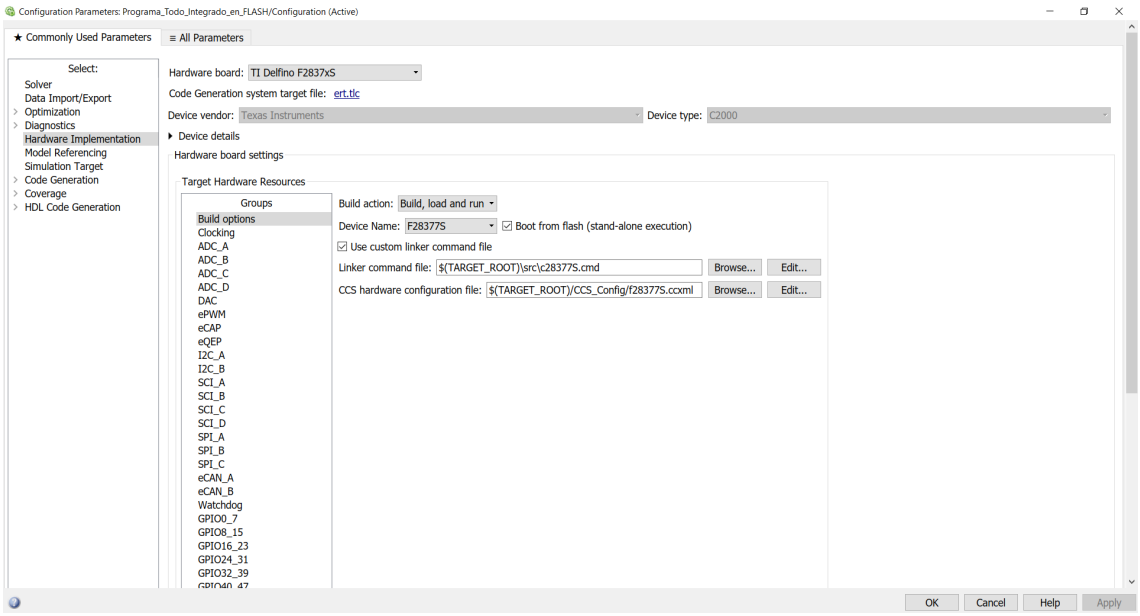


Figura 3-9. Configuración del proyecto de Simulink. Target Hardware Resources. Build Options.

- ADC_A, ADC_B, ADC_C.
  - ADC Clock prescales (ADCCLK): SYSCLKOUT/5.0.

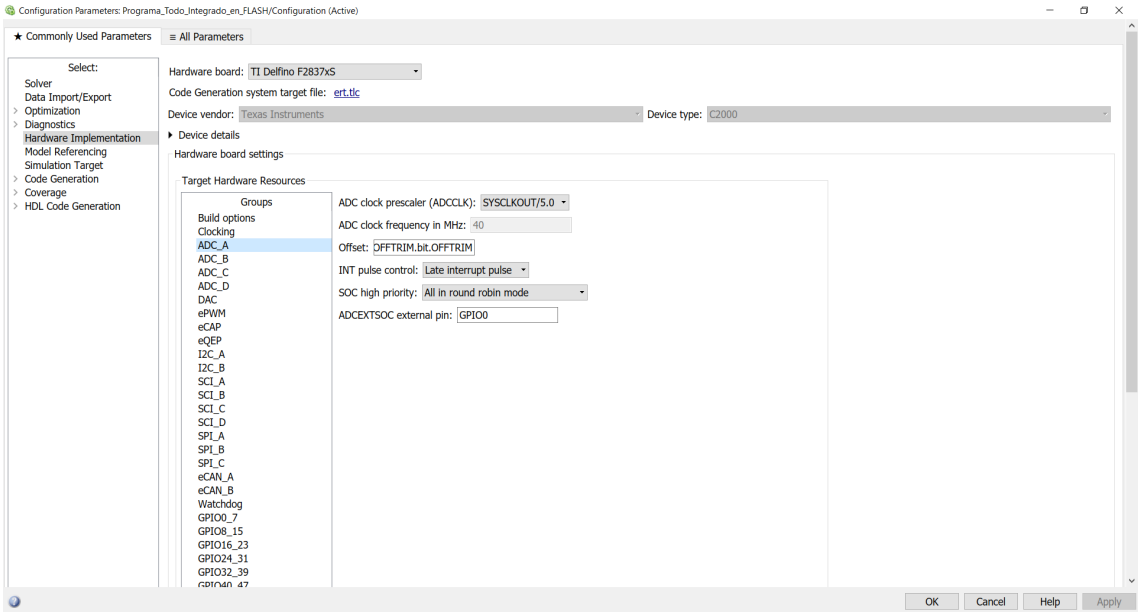
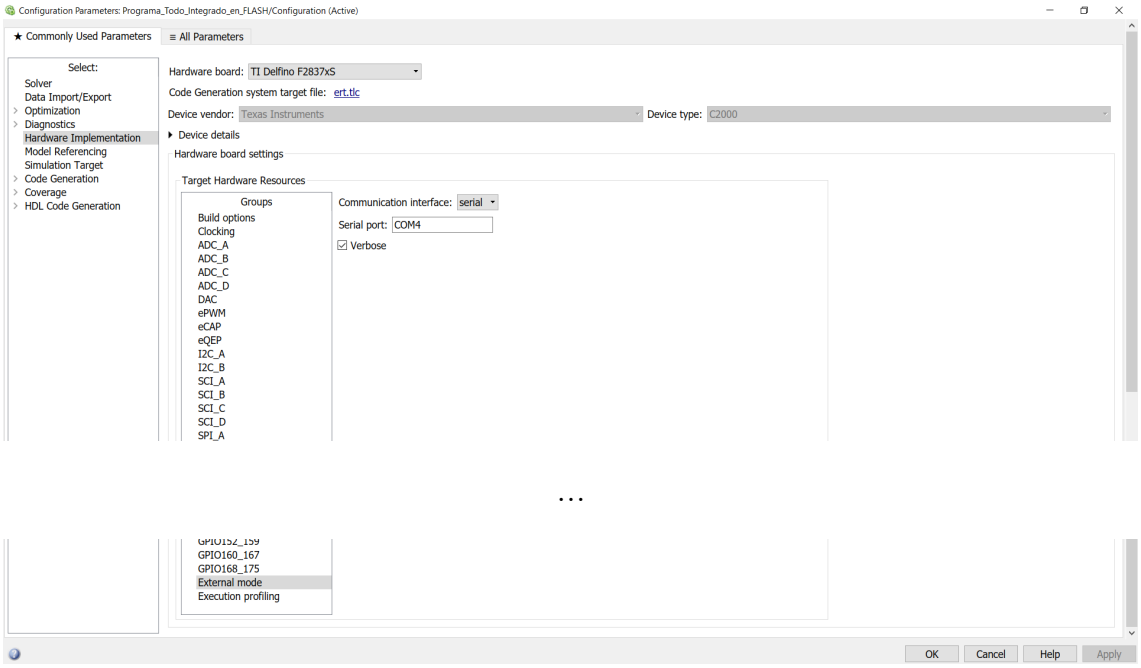


Figura 3-10. Configuración del proyecto de Simulink. Target Hardware Resources. Configuración ADCx.

- External mode.
    - Communication interface: serial.
    - Serial port: COMX*
    - Activar: Verbose.



...

Figura 3-11. Configuración del proyecto de Simulink. Target Hardware Resources. External Mode.

*El puerto COMX será aquel donde esté conectado el microcontrolador.

4 SIMULACIÓN DEL CIRCUITO EN SIMULINK

4.1. Circuito Virtual del Elevador.

Como paso previo a la implementación física del proyecto, se ha realizado un proyecto de Simulink, donde se va a poner en marcha un Convertidor Elevador. Con este circuito se pretende comprobar el comportamiento esperado del circuito donde, además, se va a diseñar el control en buble cerrado paso a paso.

Con el diseño que se va a implementar, el circuito parte de una tensión en los bornes de entrada de 100 V, que será elevada a 200 V en los bornes de salida. La frecuencia con la que se programa el bloque 'powergui' de Matlab, que permite analizar y medir parámetros eléctricos, como voltaje o corriente será de 1 MHz.

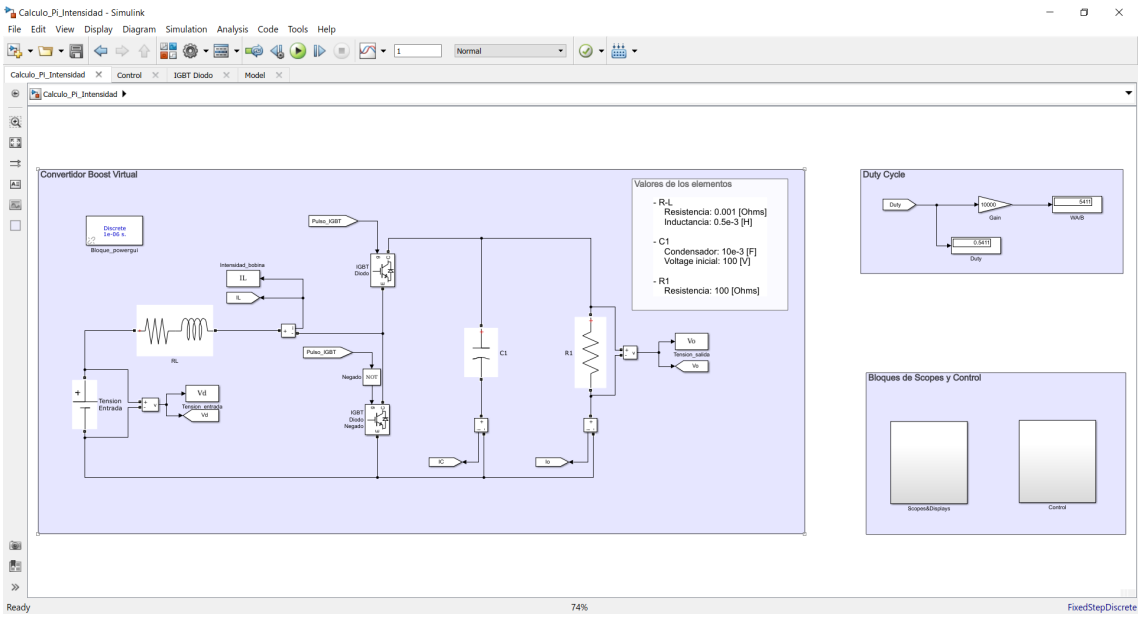


Figura 4-1. Circuito Elevador en Simulink.

Sean los elementos del circuito:

$V_d\ [V]$	$V_o\ [V]$	f powergui $[Hz]$	$R_o[Ohms]$
100	200	1.000.000	100
RL		C	
$0.001\ [Ohms]$ $0.5\ [H]$		$10e^{-3}\ [F]$ $V_{ini} = 100\ [V]$	

Tabla 4-1. Valores de los componentes de la simulación realizada en Simulink.

Los interruptores IGBT han sido configurados de la siguiente forma:

- Internal resistance: $R_{on} = 1e^{-3}\ Ohms$.
- Snubber resisance: $R_s = 1e^{5}\ Ohms$.
- Snubber capacitance: $Cs = inf\ F$.

En el circuito real se programarán los ADC de forma que recojan datos cada 8 kHz, velocidad a la que se realicen los cálculos de control. Esta toma de medidas no recoge todos los valores del circuito, ya que este opera a una velocidad mayor.

Se definirá por esto la velocidad del circuito virtual a $1MHz$. Se implementa con un bloque powergui configurado de la forma:

- Simulation type: Discrete.
- Sample time (s): $1e^{-6}$.

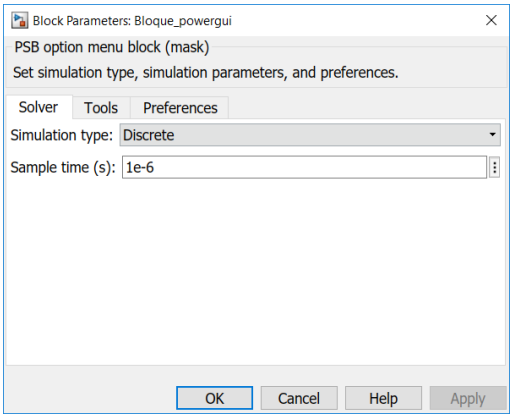


Figura 4-2. Configuración del bloque powergui de la simulación.

El subsistema "Scopes and displays", será donde se recopilarán las medidas del programa:

- Tensión de entrada, V_d $[V]$.
- Tensión de salida, V_o $[V]$.
- Intensidad por la bobina, I_L $[A]$. Además de I_c, I_o para comprobar el comportamiento.
- Duty Cycle (pulsos del IGBT e IGBT negado).

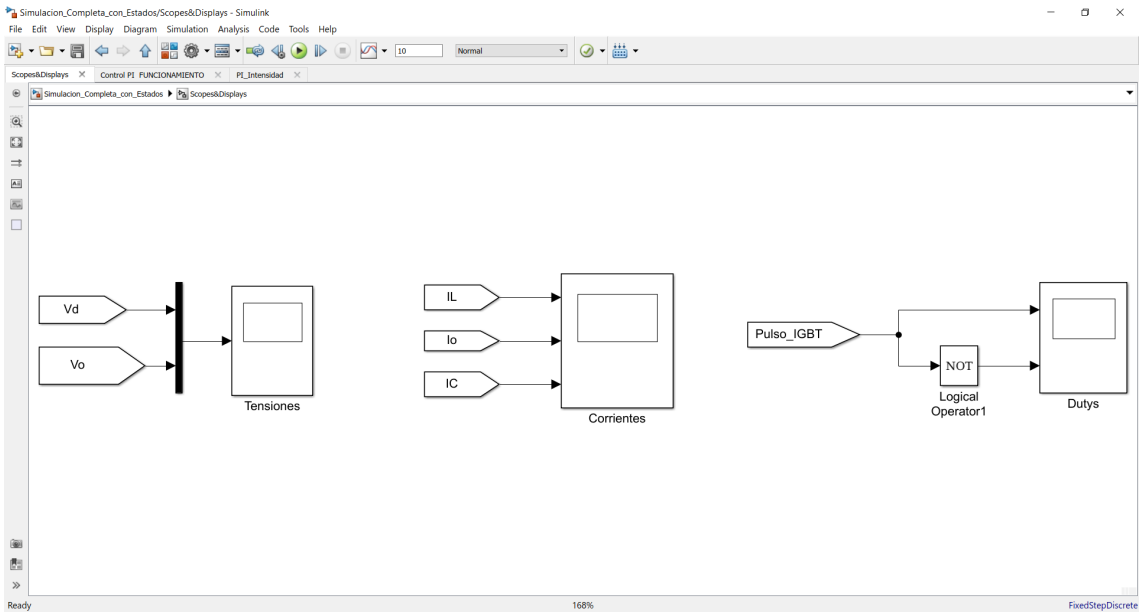


Figura 4-3. Scopes preparados para representar las variables deseadas.

4.2. Control en bucle cerrado.

El diagrama de control que se pretende aplicar al circuito está formado por dos bloques PI.

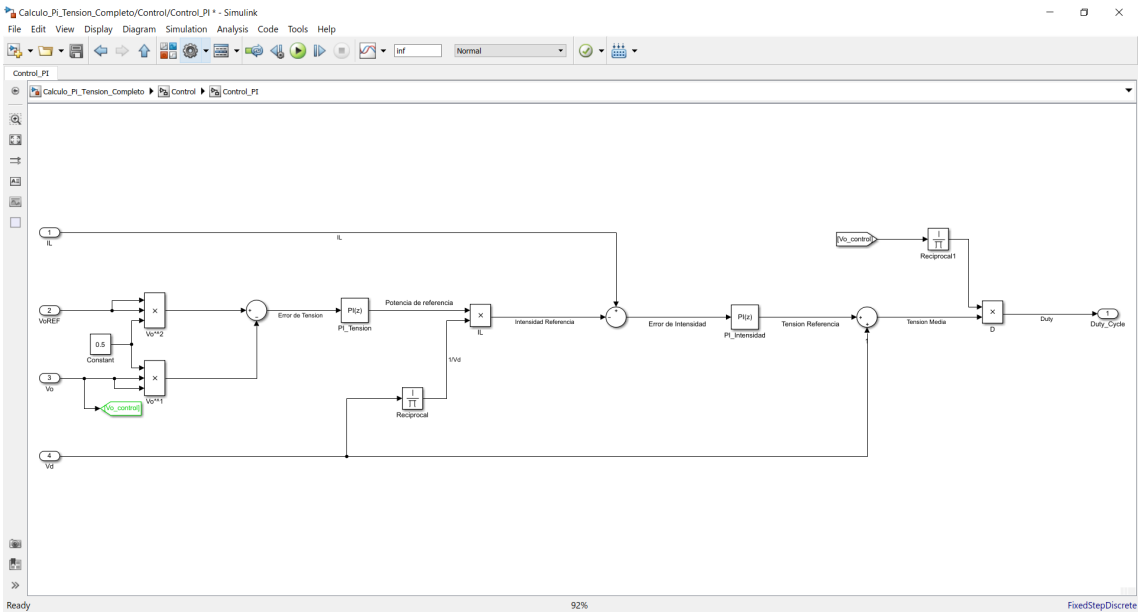


Figura 4-4. Diagrama de control del convertidor.

Se crea una etapa adaptadora, al ejecutarse a distinta frecuencia el control del circuito con “Rate Transitions”.

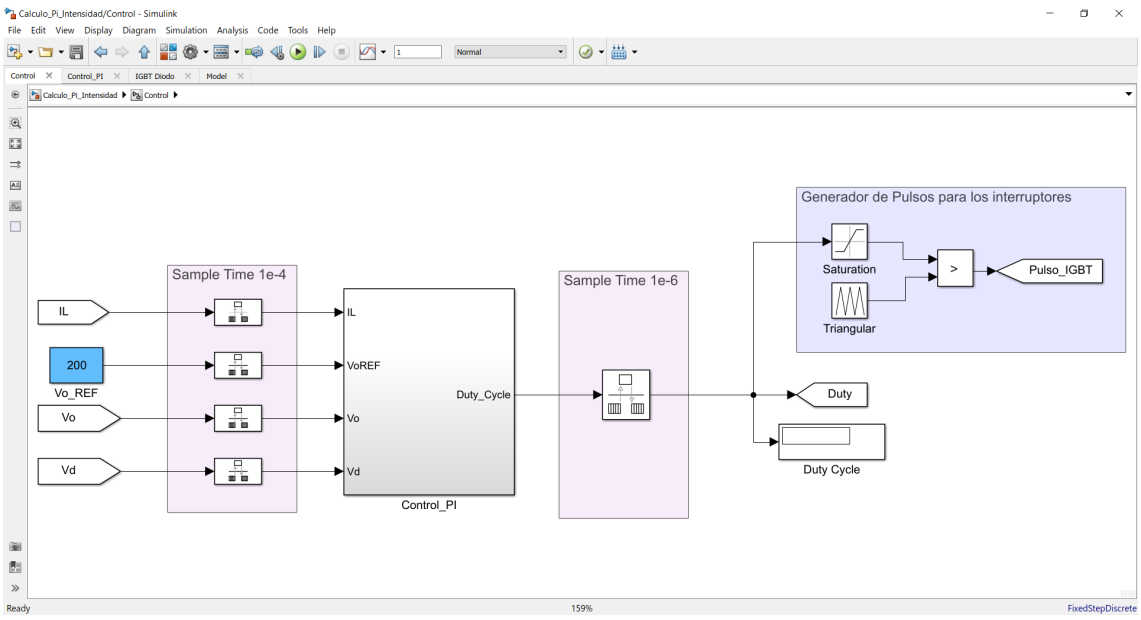


Figura 4-5. Circuito adaptador de las medidas V_o, V_d, I_L, y $Duty\ Cycle$.

Sin embargo, hay que diseñar los valores de estos Controladores PI (Proporcional Integral).

En primer lugar, se hace funcionar el circuito en bucle abierto, calculando el PI de Intensidad.

Calculada la respuesta del controlador, cerramos el lazo y se calcula el Controlador PI de tensión.

4.2.1 Cálculo de los parámetros del Controlador PI de intensidad en bucle abierto.

Se estima una potencia de referencia, representada como una constante en Simulink.

$$P_{ref} = \frac{V_o^2}{R} = \frac{200^2}{100} = 400\ W$$

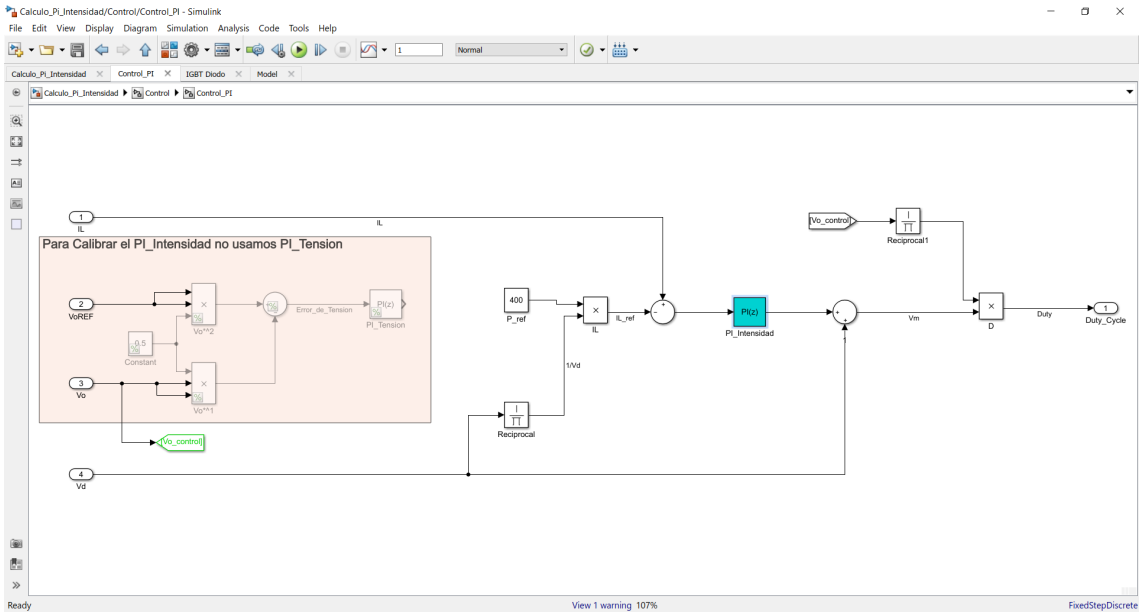


Figura 4-6. Bloque de control para determinar las constantes del PI de Intensidad.

Además de esto, en el circuito del elevador hay que sustituir el condensador por una fuente de tensión de la misma tensión que se quiere tener a la salida del circuito.

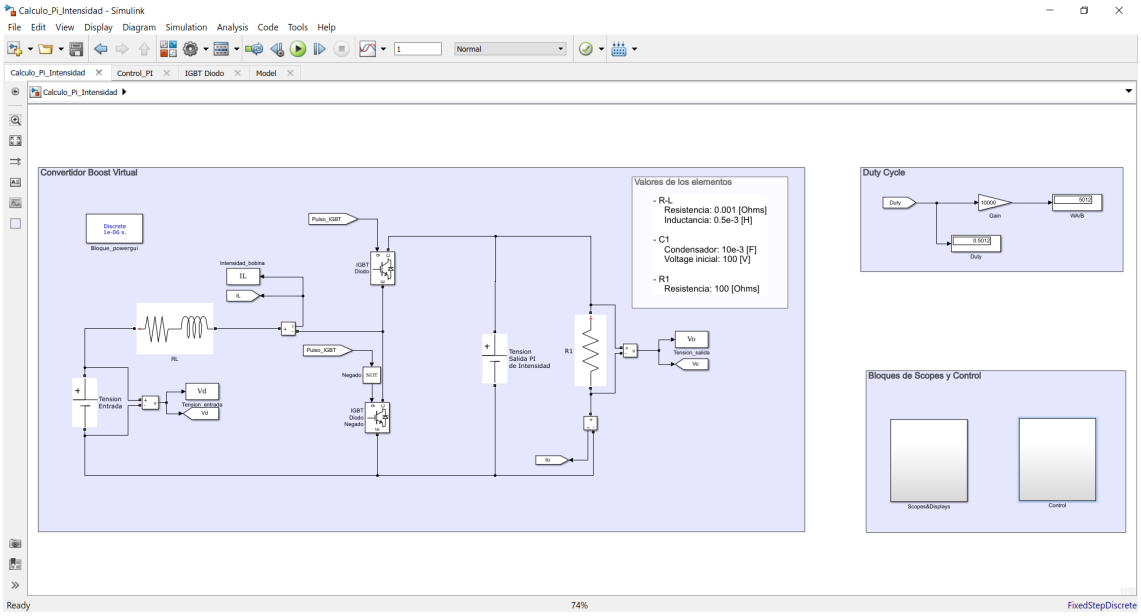


Figura 4-7. Circuito elevador modificado para determinar las constantes del PI de Intensidad.

Se busca alcanzar un bucle rápido, así que hay que procurar que la respuesta del controlador sea lo más rápida posible, y que no produzca grandes oscilaciones.

Parámetro Controlador Intensidad.	
K_P	3
K_I	100

Tabla 4-2. Parámetros del controlador PI de intensidad.

Comprobando que los resultados son correctos, la Intensidad de referencia será:

$$I_{ref} = \frac{P_{ref}}{V_d} = \frac{400}{100} = 4\ A$$

Obteniendo las gráficas de la intensidad del circuito, se observa que su valor medio es 4 A.

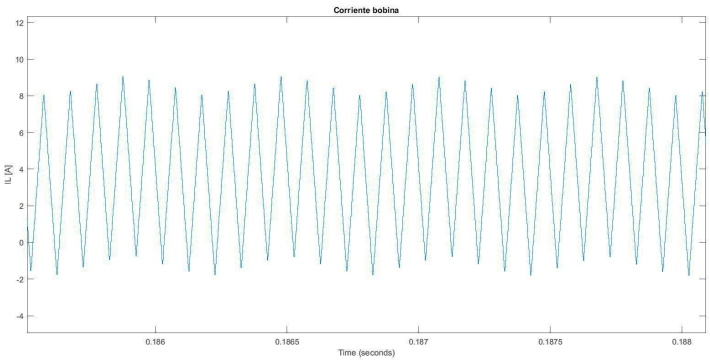


Figura 4-8. Valor de la Intensidad por la bobina de la simulación.

4.2.2. Cálculo de los parámetros del Controlador PI de tensión en bucle cerrado.

Con el primer controlador determinado, se cierra el bucle agregando el controlador de la tensión, además de restaurar el convertidor inicial cambiando la fuente de tensión, por el Condensador.

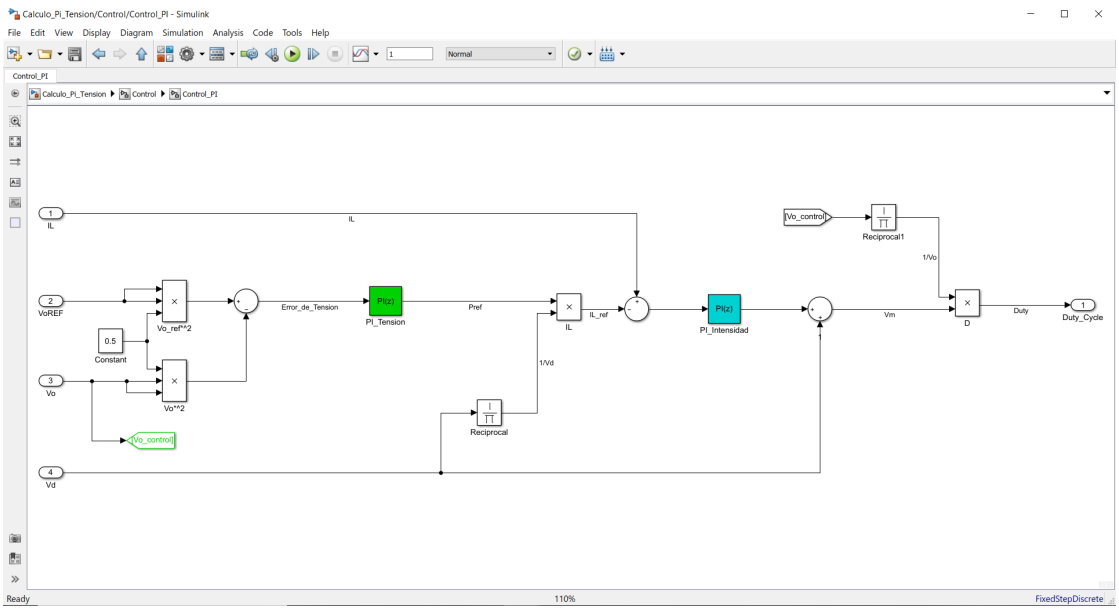


Figura 4-9. Bloque de control para determinar las constantes del PI de Tensión.

A la hora de diseñar este bloque de control, impera la necesidad de que llegue a la tensión de salida deseada sin sobreoscilaciones.

El resultado es:

Parámetros Controlador Tensión.	
K_P	1.5
K_I	5

Tabla 4-3. Parámetros del controlador PI de tensión

4.2.3. Operación del convertidor con Control Operativo.

Una vez se encuentra el bloque de control completo, con los controladores tanto de intensidad como de tensión definidos, se realizará una simulación para comprobar que la respuesta es la que se quería alcanzar.

De ésta, se espera que la tensión de salida alcance el valor de referencia impuesto en la simulación, que en el caso de esta simulación es $V_{o\,ref} = 200\,V$, sin producir grandes sobreoscilaciones y lo más rápidamente posible.

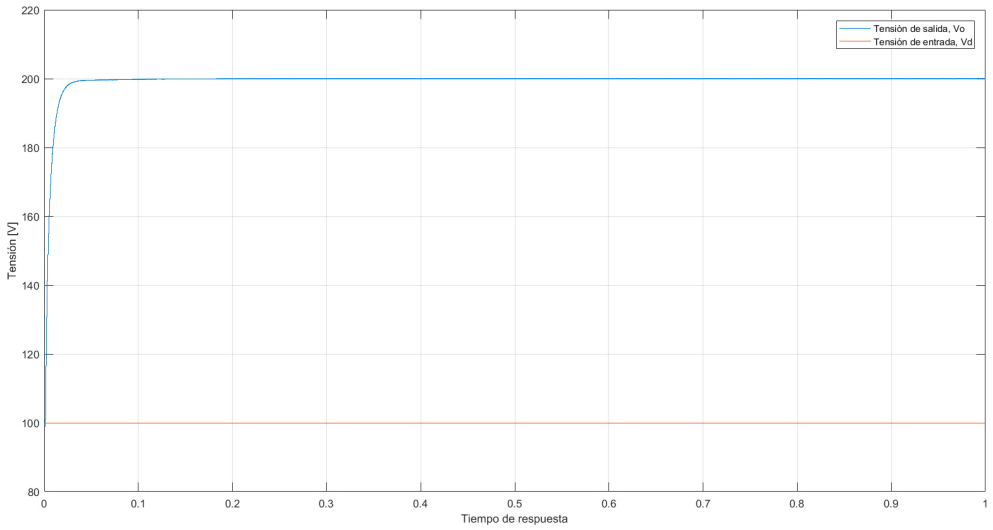


Figura 4-10. Tensión de entrada V_d, frente a tensión de salida V_o implementando control.

Se observa también que el Duty Cycle obtenido por la simulación, una vez se estabiliza la tensión de salida en $V_{o\,ref}$, alcanza un valor muy próximo al teórico:

$$D = 1 - V_d/V_o = 1 - 100/200 = 1 - 0{,}5 = 0{,}5$$

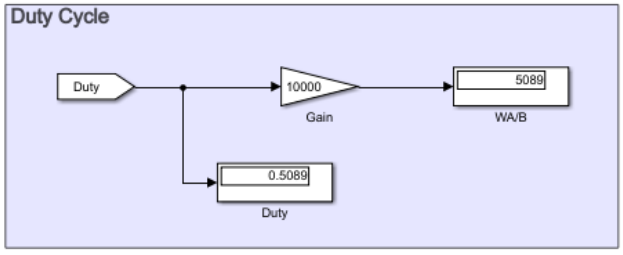


Figura 4-11. Valor del Duty Cycle obtenido por simulación aplicando bloque Control.

4.3. Implementación de una máquina de estados.

Para el funcionamiento real del convertidor será necesario implementar una serie de estados de funcionamiento, desde reposo hasta que opere correctamente. Esto se conseguirá mediante el diseño de una máquina de estados, la cual permita cambiar de un estado de funcionamiento a otro, cuando se cumplan las condiciones requeridas.

Comprobado que el control diseñado para el convertidor funciona, hay que cerciorarse de que las transiciones entre los estados del convertidor, no distorsionan la respuesta esperada.

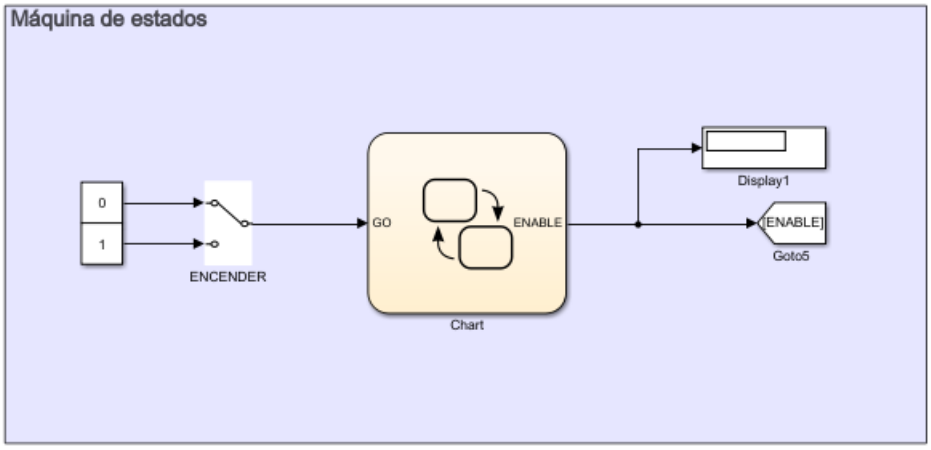


Figura 4-12. Máquina de estados del sistema.

La entrada de la máquina de estados está formada por un interruptor "switch", y dos valores fijos. Al activar el interruptor, enviará $\{0,1\}$ como variable a evaluar.

Se diseña dos estados, con los que observar la reacción al pasar de un estado de precarga, "REPOSO", donde la tensión de salida sea similar a la de la entrada ($V_o = V_d$), al estado de operación "FUNCIONAMIENTO", donde el bucle de control desarrollado previamente debe lograr que la tensión de salida alcance el valor de referencia, $V_o = 200\,V$.

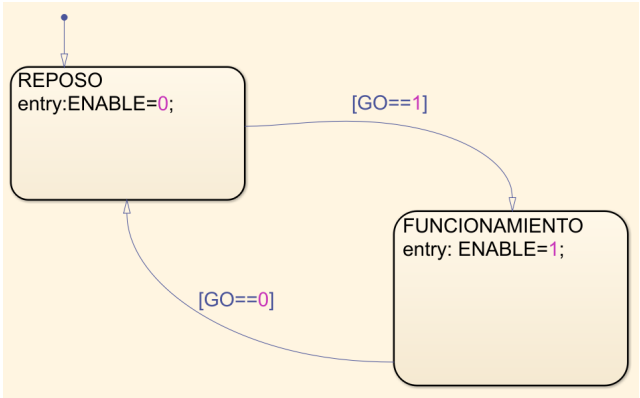


Figura 4-13. Máquina de estados del sistema. Estados declarados.

Siendo la única salida de la máquina de estados la variable "ENABLE", con la que se va a alternar entre dos valores de la tensión de referencia del controlador.

El bloque de control lo se adapta a la nueva funcionalidad del circuito, donde se han simplificado los bloques de las anteriores simulaciones en uno, para observar el comportamiento de las señales.

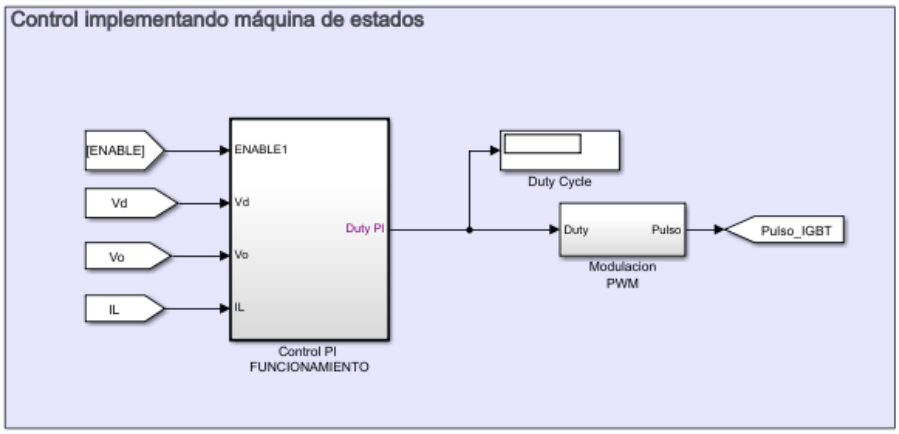


Figura 4-14. Bloque de Control implementando máquina de estados.

La señal de salida "ENABLE· de la máquina de estados va a tener dos tareas dentro de la simulación:

- Habilitará una tensión de referencia $V_{o\,ref}$ entre los valores de $\{100, 200\}\,V$. Gracias a esto, la máquina de estados emula el caso en el que $V_o = V_d$ correspondiente a "REPOSO", y $V_o = 200\,V$ correspondiente a "FUNCIONAMIENTO", poniendo a prueba el control diseñado.
- Accionamiento de un disparo, o "trigger" de reset habilitado en los controladores PI para que estos no acumulen error durante el estado de "REPOSO.

En los controladores PI se incluye la opción de "External reset", que se activarán ante una señal de "rising" de entrada. En el circuito, cuando se pase de "REPOSO" a "FUNCIONAMIENTO", la variable "ENABLE" pasará de un estado de $0 \rightarrow 1$, activando así el trigger rising, y reseteando los bloques PI.

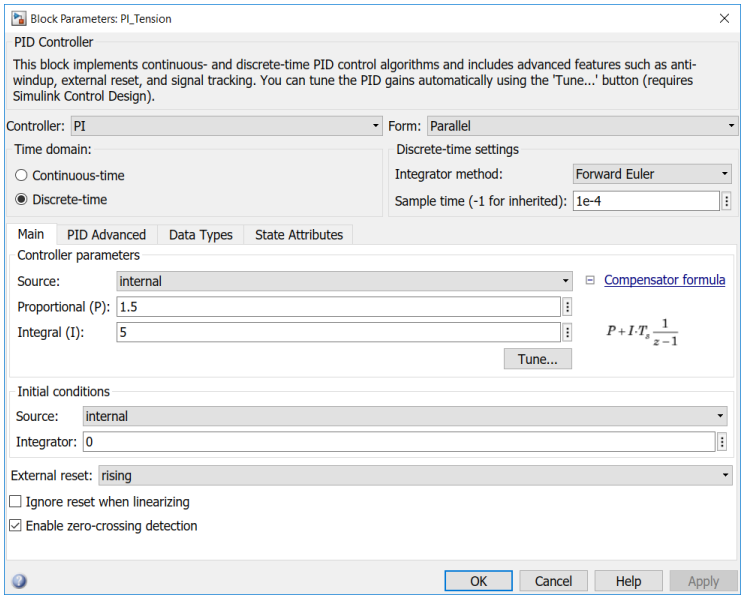


Figura 4-15. Configuración de los bloques PI para habilitar el reset.

Mostrando el bloque de control con más detalle:

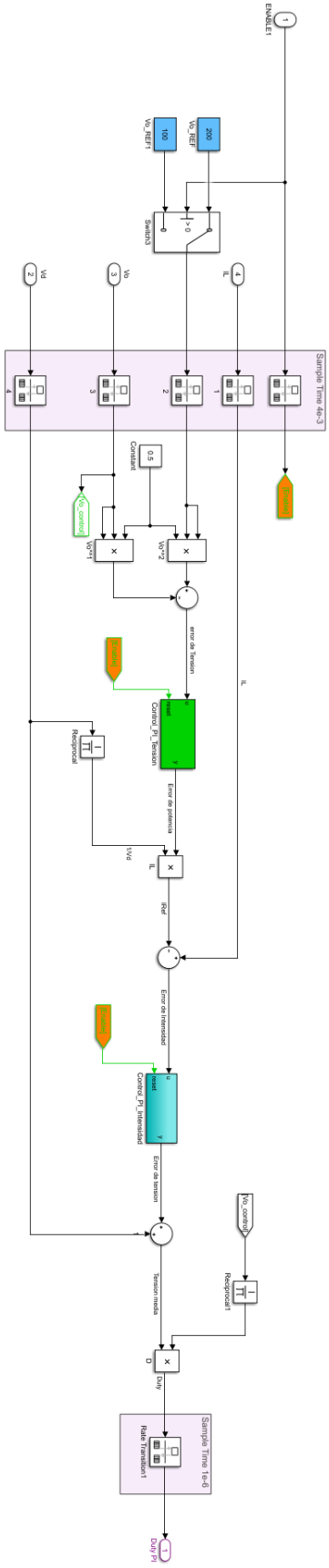


Figura 4-16. Bloque de Control con señal ENABLE.

4.4. Resultados de la simulación.

Con la máquina de estados y los bloques de control implementados, se pone en marcha el circuito, prestando especial interés en los estados de transición provocados por el uso de la máquina de estados.

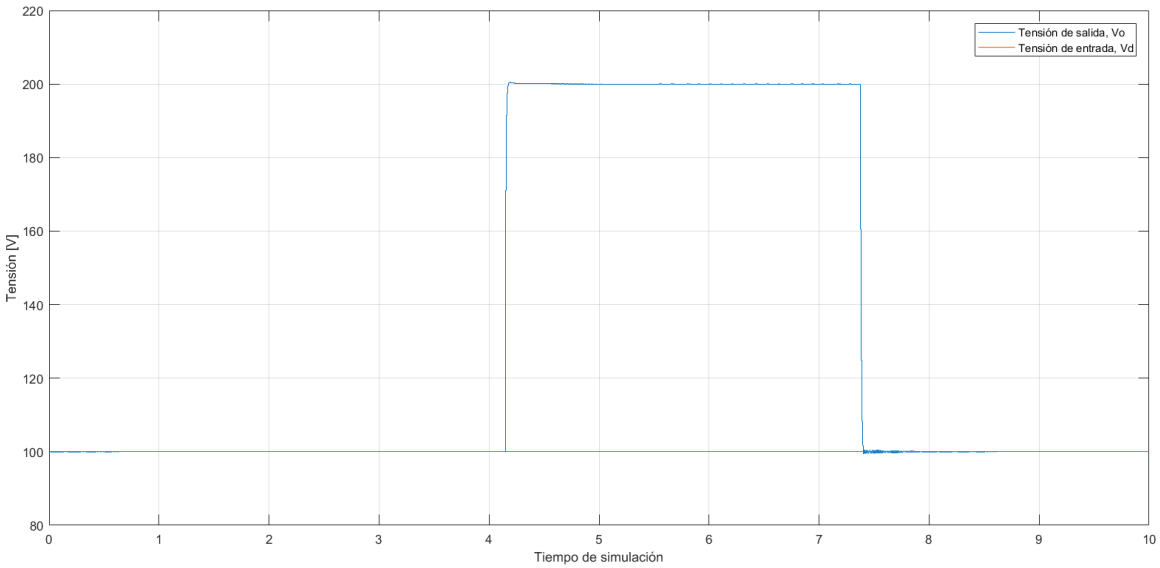


Figura 4-17. Tensiones $V_d - V_o$ durante una simulación operando la máquina de estados.

Al comienzo de la simulación, partiendo del estado de REPOSO, la tensión de salida V_o tiende a un valor muy próximo a la tensión de entrada $V_d = 100\ V$.

Al activar el interruptor ENCENDER, V_o crece rápidamente hasta alcanzar el valor de $200\ V$ impuesto por el controlador, sin provocar una sobreoscilación que ponga en peligro el equipo.

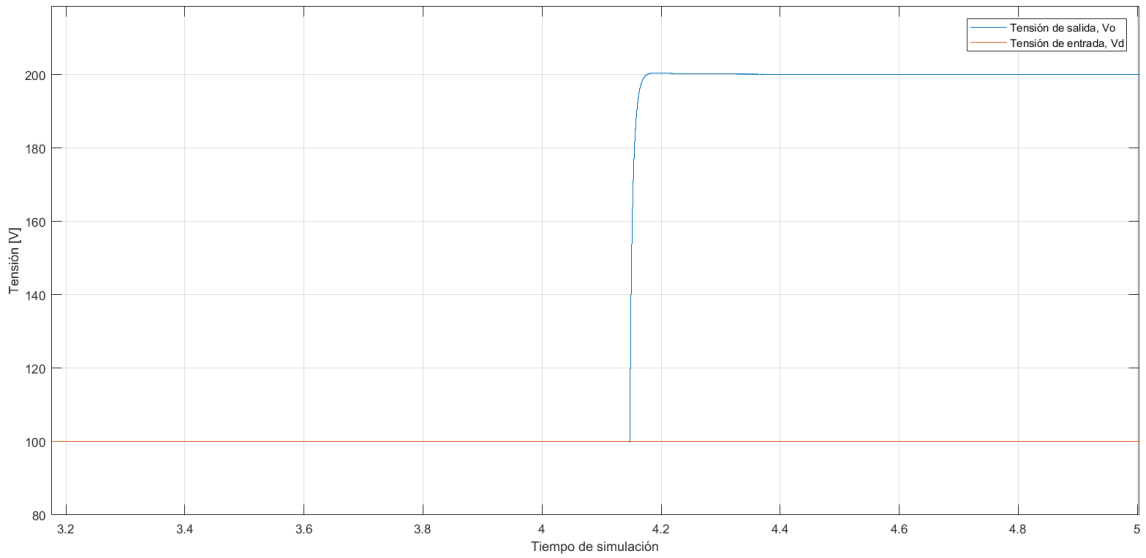


Figura 4-18. Respuesta Convertidor en transitorio de REPOSO a FUNCIONAMIENTO.

Así mismo se comprueba que el Duty calculado por el bloque Control PI FUNCIONAMIENTO oscila el valor 0.5, calculado previamente.

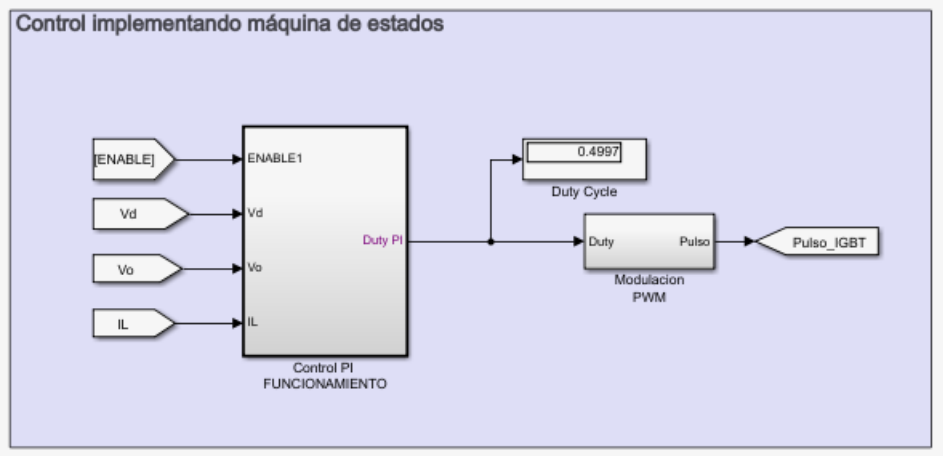


Figura 4-19. Valor del Duty Cycle con convertidor trabajando en estado FUNCIONAMIENTO.

5 PROGRAMACIÓN DEL DIAGRAMA EN SIMULINK

A la hora de desarrollar el control del convertidor de tensión, la aplicación del circuito real se ha programado un diagrama de simulink donde se aplica el control desarrollado en el apartado anterior, además de otras funcionalidades, relacionadas con los elementos externos con los que se va a trabajar.

Para tomar las medidas e interaccionar con los elementos externos del circuito, se utilizará el microcontrolador TMS320 F28377S, ya comentado previamente, el cual ejecutará el siguiente programa diseñado mediante la herramienta Simulink de Matlab.

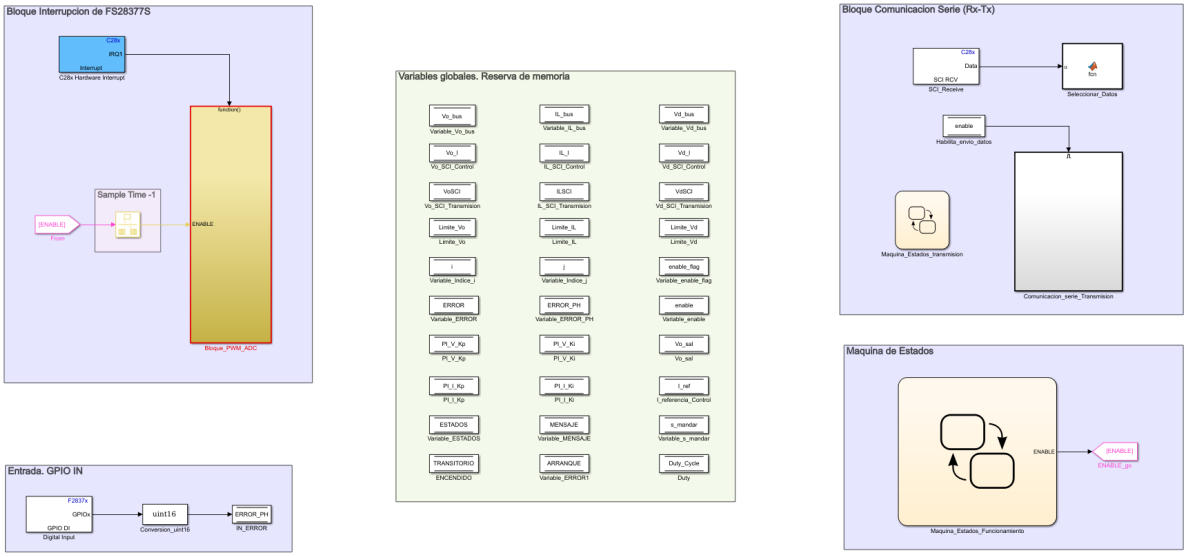


Figura 5-1. Diagrama Simulink Convertidor real. Programa completo.

A simple vista se pueden diferenciar 5 zonas, las cuales se van a desarrollar de aquí en adelante.

- Bloque de Interrupción F28377S.
- GPIO IN.
- Bloque Comunicación Serie (Rx-Tx).
- Variables globales. Reserva de memoria.
- Máquina de estados.

5.1. Bloque de Interrupción F28377S.

El modo de operación del sistema requiere de la programación de un sistema de interrupción, habilitada por la interrupción de los canales ADC.

¿Cómo se va a configurar esta interrupción?

La interrupción del sistema va a venir implementada mediante un módulo PWM y un ADC del microcontrolador de la siguiente forma.

- Se habilita un PWM del microcontrolador, y se activa un disparador de evento "event trigger", que inicie la conversión de un ADC.
- En la configuración del ADC, se define como fuente de disparo "trigger source" el módulo PWM que se ha habilitado para la conversión. Además, se habilita una interrupción de ADC, ADCINTx para cuando acabe de convertir el módulo ADC.

De esta forma la interrupción se habilita cada vez que se inicia un periodo del PWM, se trata la información obtenida de los módulos ADC para aplicar el control pertinente, hasta el siguiente periodo de la señal PWM.

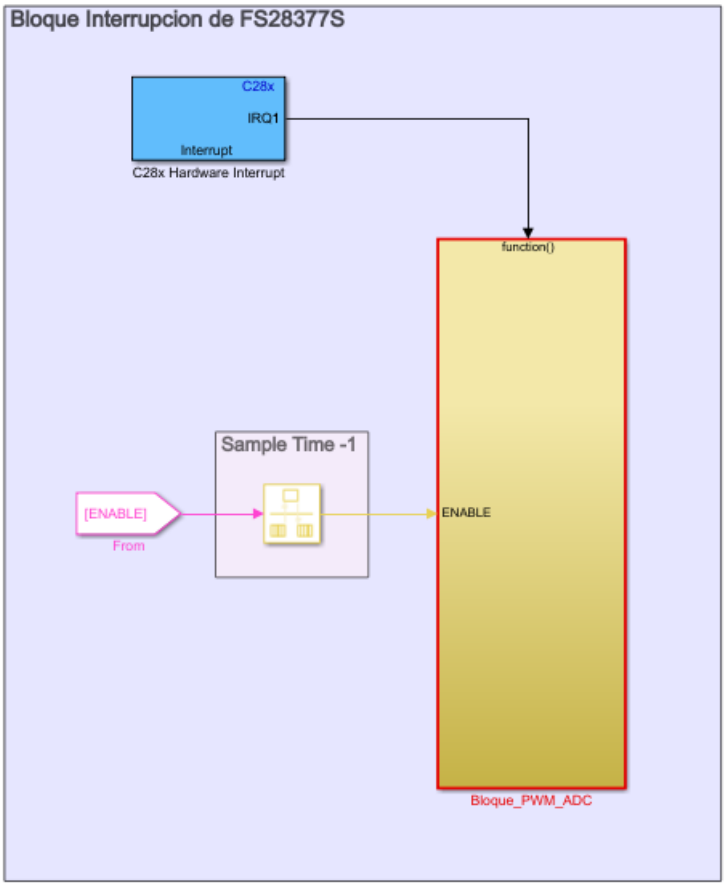


Figura 5-2. Diagrama Simulink Convertidor real. Área de Interrupción.

5.1.1. Bloque Hardware Interrupt.

Este bloque define qué tipo de interrupción se instala en el programa.

Puesto que sólo se necesita habilitar la interrupción de un módulo ADC, más concretamente del módulo ADCINT1, basta con implementar esta interrupción en el bloque.

La prioridad de interrupción no es relevante, al programar sólo una interrupción.

CPU Interrupt numbers	[1]
PIE Interrupt numbers	SEQ1INT (ADC) / ADCINT1/ ADCA1
[1]	

Tabla 5-1. Configuración bloque Hardware Interrupt.

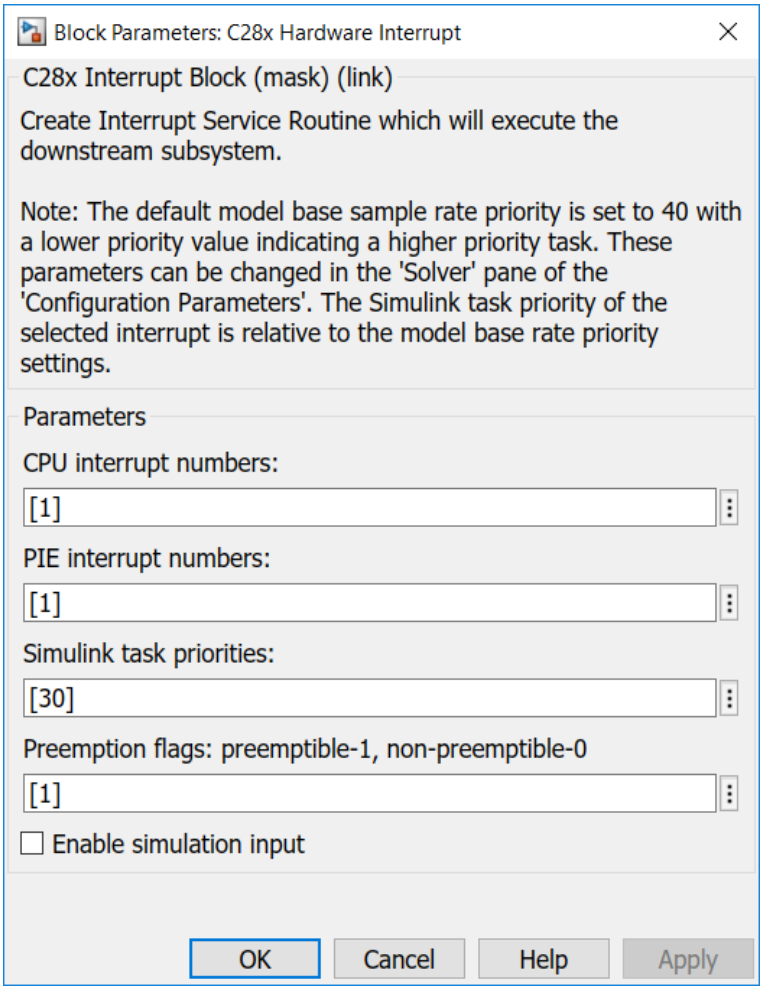


Figura 5-3. Diagrama Simulink Convertidor real. Área de Interrupción. Bloque Hardware Interrupt.

5.1.2. Bloque PWM, ADC y Control.

Dentro de este bloque se va a programar todo lo relativo al módulo PWM, los tres ADC cuya misión será la de tomar las medidas que se necesitan del circuito real, y el control desarrollado en el apartado 4 del proyecto.

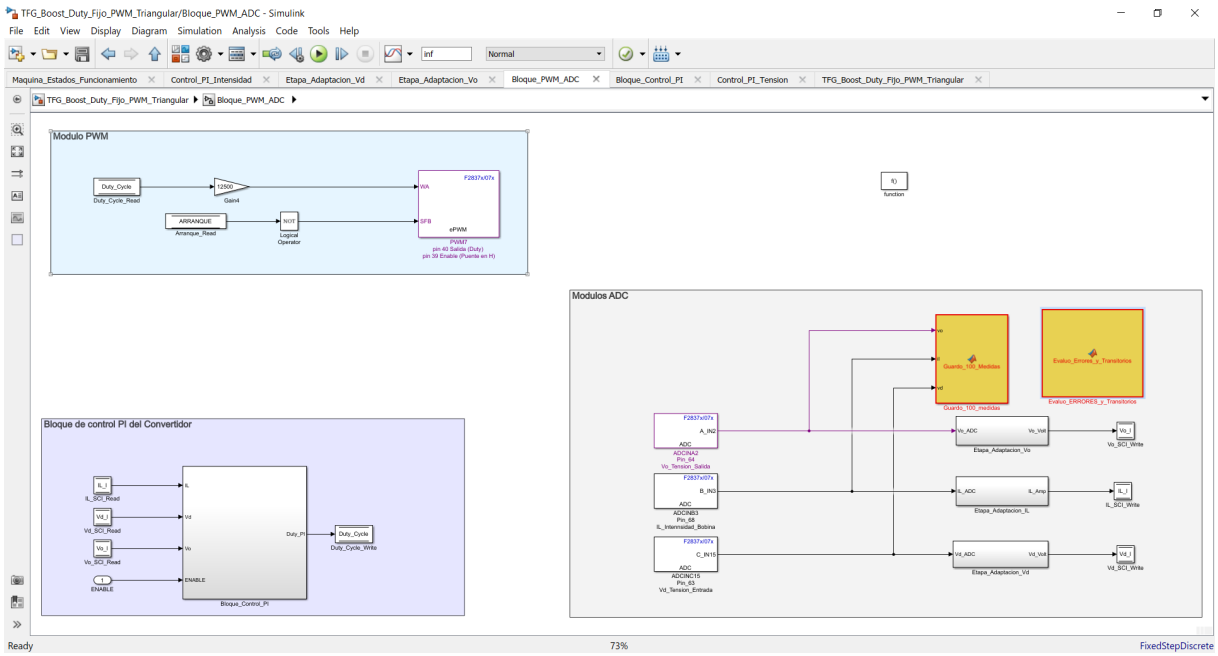


Figura 5-4. Diagrama Simulink Convertidor real. Área de Interrupción. Bloque PWM, ADC y Control.

5.1.3. Programación del módulo PWM.

Para el control de los interruptores del puente H, que corresponden a los transistores IGBT de la simulación virtual, es necesario generar una señal de modulación, que implemente un Duty Cycle requerido por el sistema, una vez sea calculado por el bloque de control.

Se usará el módulo PWM7, cuyos módulos A/B corresponden a los pines:

Módulo PWM7.	Pin.
PWM7A	40
PWM7B	39

Tabla 5-2. Correspondencia de módulos PWM7 con pines del launchpad.

En primer lugar, para la configuración el módulo PWM hay que saber para qué periodo de ejecución se pretende diseñarlo. Puesto que el carácter predominante de la interrupción radica en la toma de datos de los ADC, habrá que calcularlo de forma que el tiempo de adquisición, así como de la implementación de control sea óptimo.

La bobina del circuito real opera a una frecuencia de 4 kHz, así que esa será la frecuencia de operación del PWM.

¿Qué tipo de PWM se va a programar?

Puesto que el PWM va a ser el trigger de los módulos ADC, hay que programarlo de forma que se tomen las medidas deseadas.

- Las tensiones V_o y V_d, puesto que tienen un valor estable durante el periodo del PWM, no requieren de ninguna configuración especial.
- La intensidad I_L si necesita de una configuración en particular, puesto es una onda triangular.

La siguiente tabla muestra la comparativa en la toma de medidas de la intensidad, usando dos modos del PWM, Up-mode y Up-down mode.

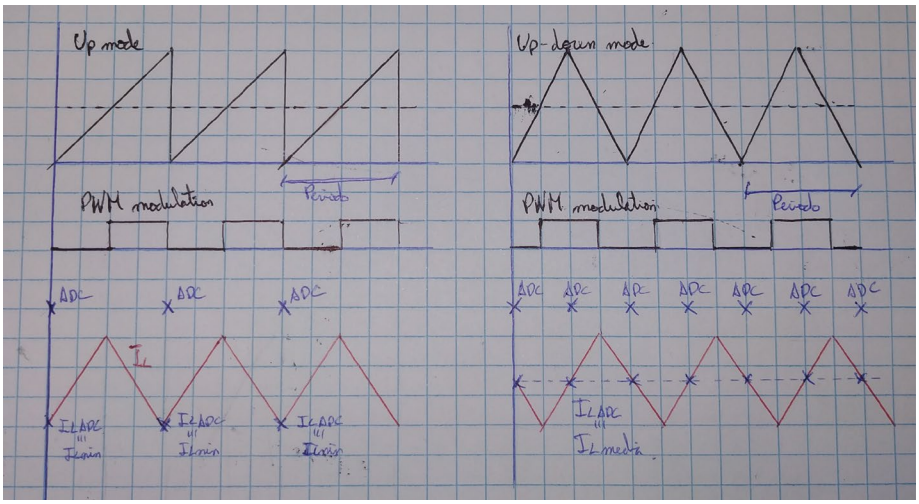


Figura 5-5. Diagrama Simulink Convertidor real. Área de Interrupción. Tabla comparativa modos PWM.

Se observa que con el PWM modulando en Up mode, los módulos ADC están tomando el valor de la intensidad mínimo de la bobina, lo cual no es apto para el control del convertidor.

Sin embargo, con el Up-Down mode, el módulo ADC está tomando los valores medios de la onda triangular que forma la Intensidad de la bobina, que son las medidas que necesita el control diseñado.

Para calcular el periodo de ejecución del PWM se procede a ejecutar con los siguientes pasos, teniendo en cuenta que el periodo del Up-Down mode, es el siguiente, especificado por el datasheet del micro.

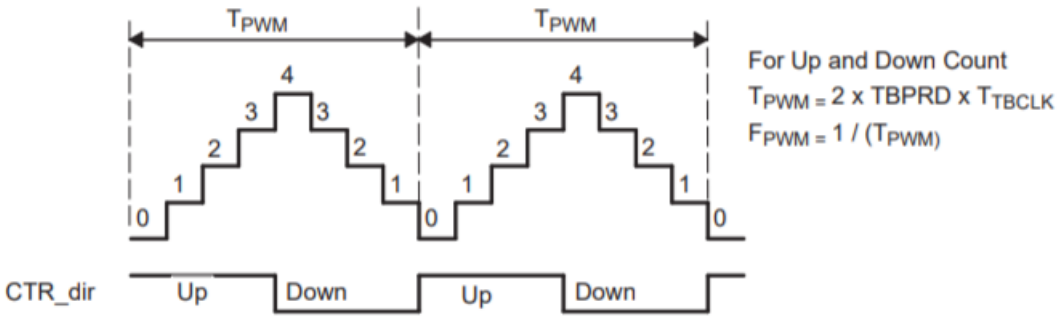


Figura 5-6. Diagrama Simulink Convertidor real. Área de Interrupción. Periodo del Up-Down mode.

1. Se obtiene la frecuencia del reloj del sistema.

$$SYSCLCKOUT = 200\ MHz$$

2. Cálculo de la frecuencia del reloj PWM.

$$PWMCLK = \frac{SYSCLKOUT}{2} = 100\ MHz$$

3. Por último, se calcula la frecuencia de ejecución del PWM, dependiendo de la variable $timer\ period$. En este caso se ha impuesto 10.000 ciclos de reloj (PWM).

$$Frecuencia\ del\ PWM7 = \frac{PWMCLK}{timer\ period * 2} = \frac{100\ MHz}{12.500 * 2} = 4\ kHz$$

Tras los cálculos, el periodo de ejecución del PWM7 es de $4\ kHz$.

Esto implica, con el modo Up-down programado, que los módulos ADC van a tomar una medida de sus canales, que corresponden con las tensiones e intensidad V_o, V_d, I_L dos veces por periodo, esto es, cada $0.125\ ms$, equivalente a $8\ kHz$.

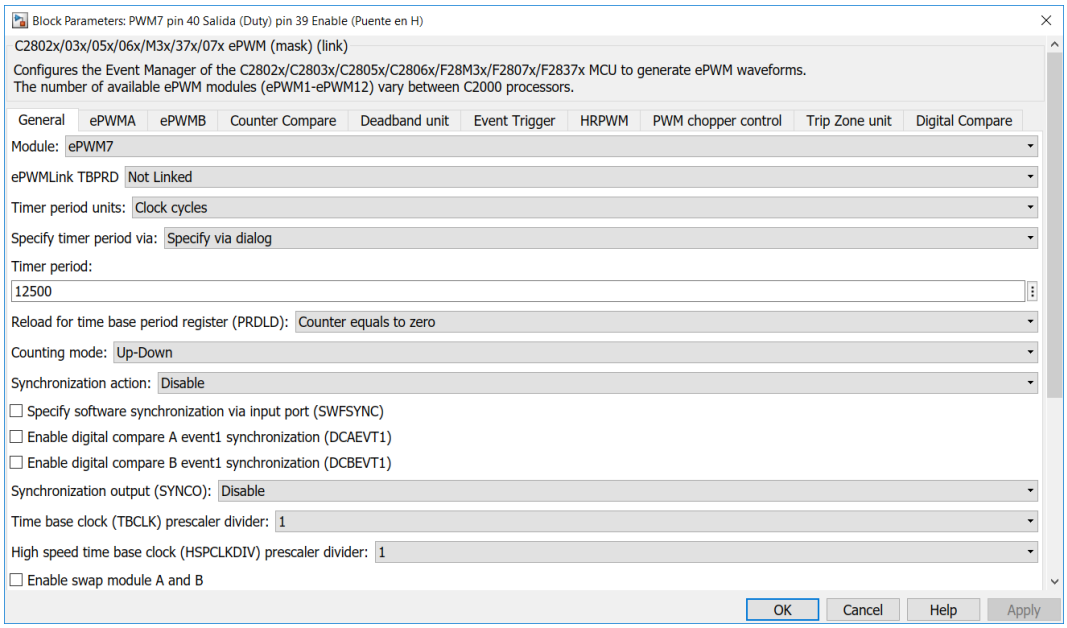


Figura 5-7. Diagrama Simulink Convertidor real. Área de Interrupción. Bloque PWM General.

Al ser necesarias dos señales, se ha habilitado los dos módulos A y B del PWM7.

PWM7A. Será el encargado de modular la señal que controle el Elevador.

- Cuando el contador sea cero, la acción a ejecutar será SET. Esto pone la salida del PWM7 a 3,3 V.
- Cuando el contador esté al mismo valor que CMPA, en la cresta creciente (up-count), se ejecuta la acción CLEAR, con lo cual la salida del PWM7A pasa a ser de 0V.
- Cuando el contador esté al mismo valor que CMPA, en la cresta decreciente (down-count), se ejecuta la acción SET, con lo cual la salida del PWM7A pasa a ser de 0V.

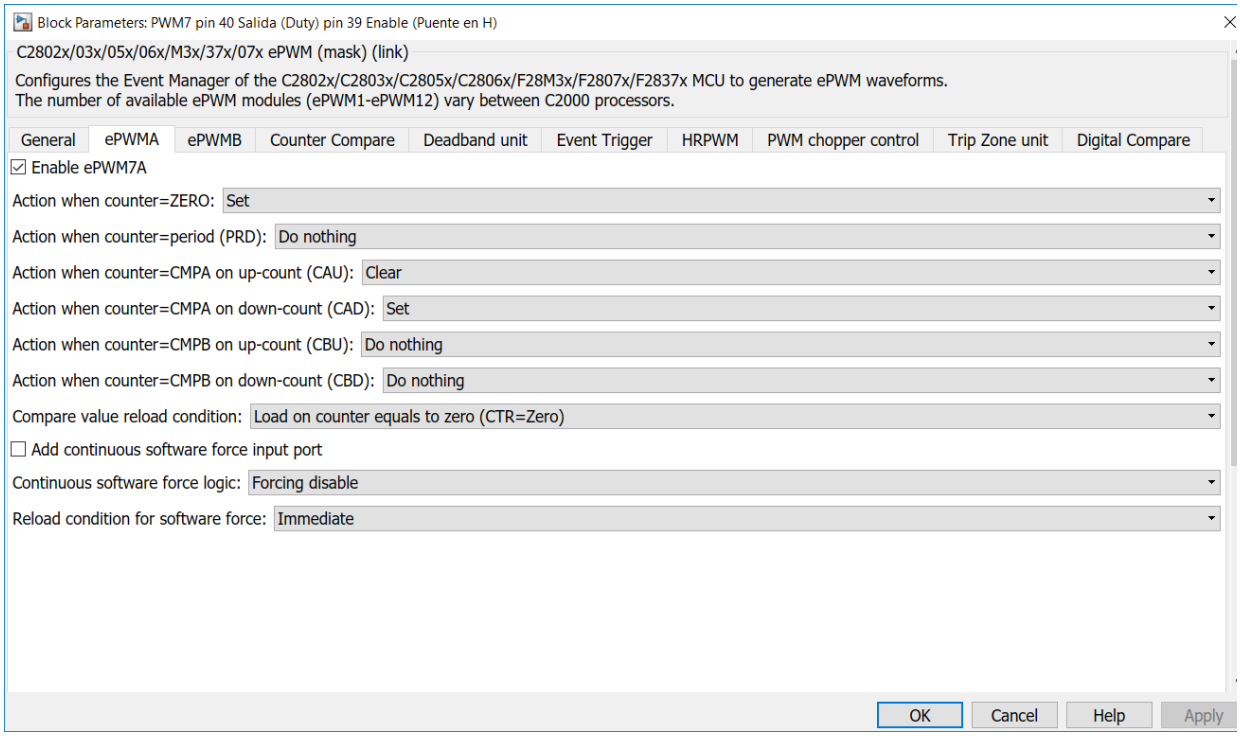


Figura 5-8. Diagrama Simulink Convertidor real. Área de Interrupción. Configuración PWM7A (1).

El valor de CMPA viene determinado via entrada en el módulo, la cual secuenta en ciclos de reloj del PWM. No es relevante el valor inicial, ya que cuando el control comience a funcionar va a ser este el que imponga su valor, a través de la entrada habilitada.

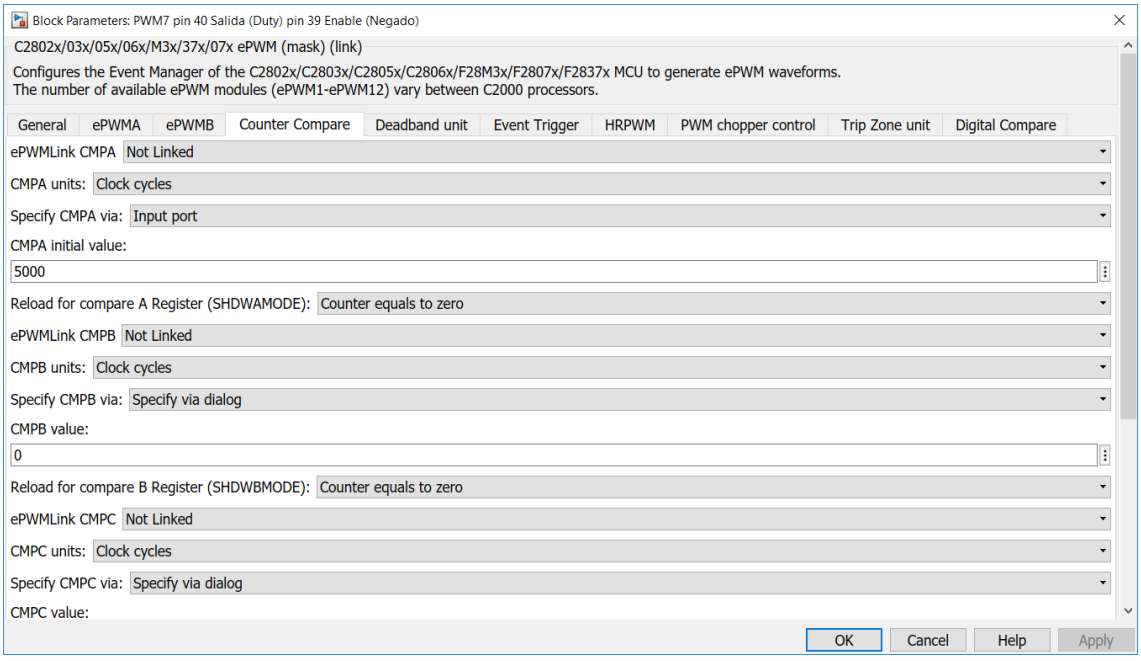


Figura 5-9. Diagrama Simulink Convertidor real. Área de Interrupción. Configuración PWM7A (2).

La entrada de este valor es el Duty, cuyo valor oscila entre [0,1], multiplicado por una ganancia de valor 12.500, con lo cual queda totalmente definido el comportamiento del PWM7A.

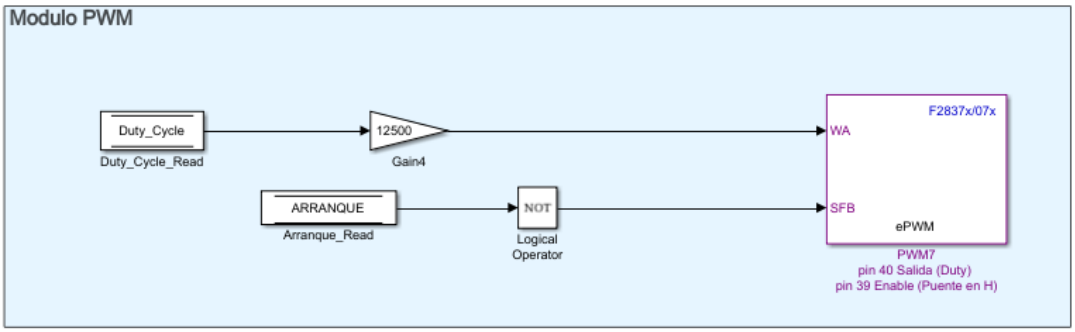


Figura 5-10. Diagrama Simulink Convertidor real. Área de Interrupción. Conversión Duty Cycle.

Por último, se habilita la interrupción de ADC, cuando CTR esté en ZERO (inicio del periodo), en PRD (final del periodo), con lo que se estará tomando dos medidas cada ciclo del PWM.

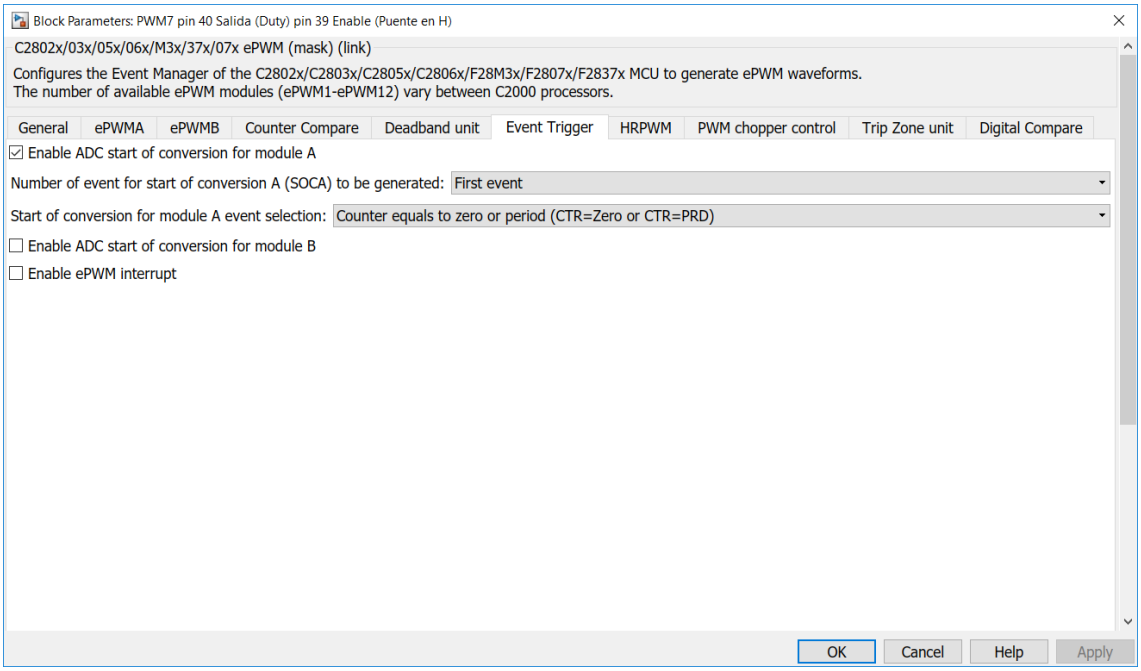


Figura 5-11. Diagrama Simulink Convertidor real. Área de Interrupción. Configuración PWM7A (3).

PWM7B. Debido al circuito del puente H, es necesaria, además de la señal modulada del PWMA, una segunda señal que se envíe como "enable" del circuito. Esto quiere decir que siempre esté a valor '1', cuando se quiera que el puente H esté operativo.

Por las características del circuito de los transmisores y receptores de fibra óptica, la señal que pasa por los bloques de Recepción de la fibra óptica, sufre una inversión de su valor debido a una etapa amplificadora.

La señal a tratar, tendrá que ser enviada convertida en su negada, ya que durante la transmisión de la información el amplificador operacional presente en el Rx de la fibra, invertirá la señal, convirtiendo un '0' en un '1' y viceversa.

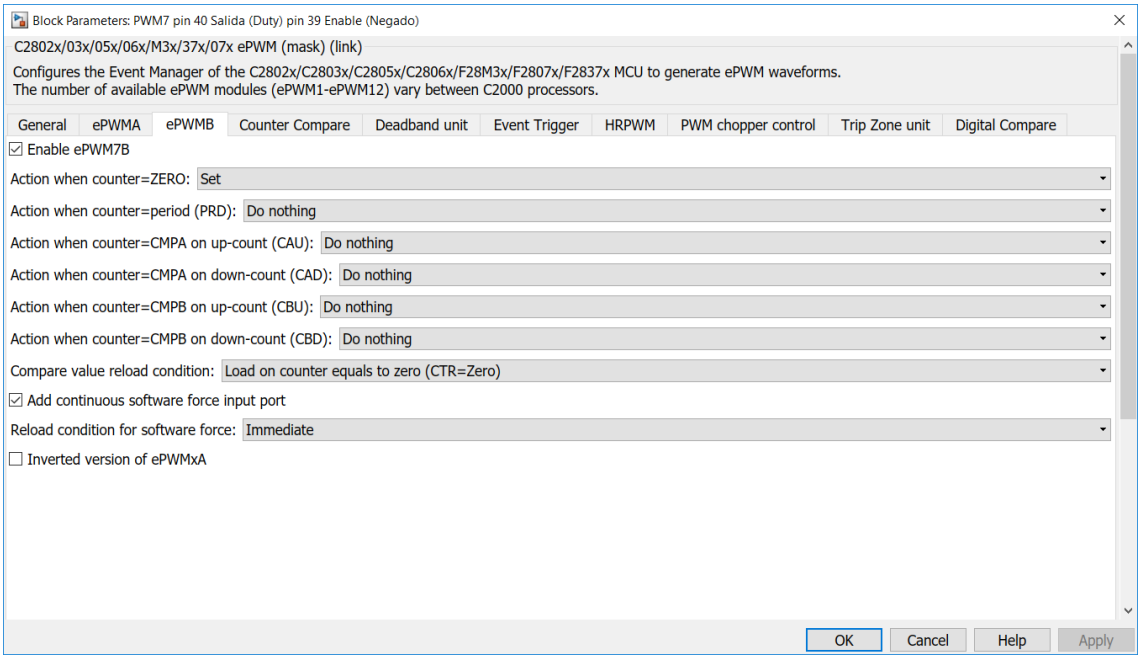


Figura 5-12. Diagrama Simulink Convertidor real. Área de Interrupción. Configuración PWM7B.

Configurados los dós módulos del PWM7, se habilita una opción que permita apagar ambos módulos del PWM, cuando no se quiera que module ninguna de las señales:

- Se activa la opción "Add continuous software force input port".
- En las opciones "Reload condition for software force: se escoge la opción "Immediate".

Al activarlo, se deshabilita del módulo PWM, lo cual es necesario cuando no sea necesaria su modulación.

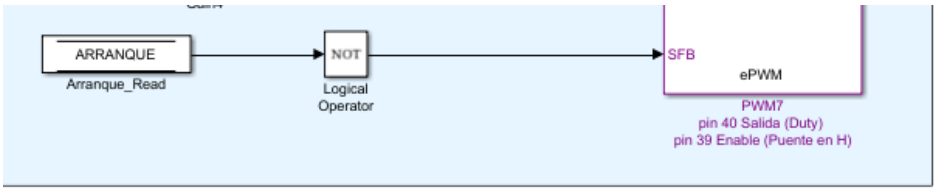


Figura 5-13. Diagrama Simulink Convertidor real. Área de Interrupción. Configuración "Software force input port (SFB)".

La variable global "ARRANQUE" será la que permita operar al módulo PWM7.

Cuando ARRANQUE vale 0 el bloque de negación convierte la señal a 1, valor que llega a SFB, desactivando el módulo PWM.

Si ARRANQUE vale 1, el parámetro SFB está ahora desactivado, puesto que ve un 0, habilitando la modulación de los dos módulos del PWM7.

5.1.4. Programación de los módulos ADC.

Puesto que el circuito necesita medir tres magnitudes distintas durante su funcionamiento, se procede a habilitar 3 canales ADC de la tarjeta TMSF28377S.

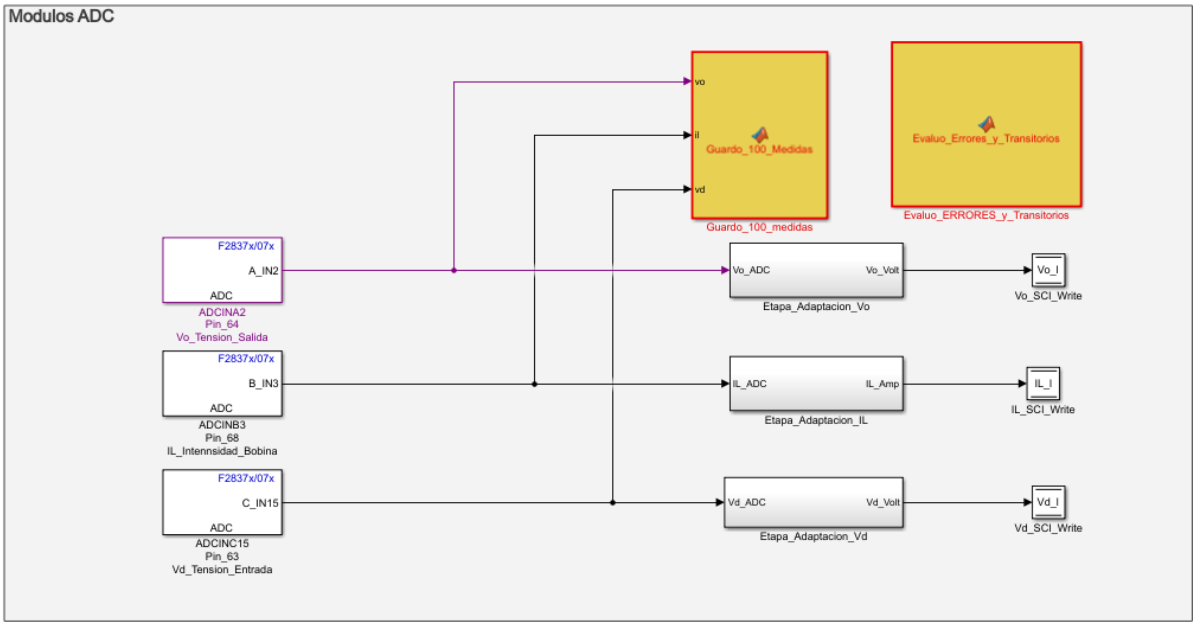


Figura 5-14. Diagrama Simulink Convertidor real. Área de Interrupción. Configuración ADC.

Los canales ADC son escogidos, de los terminales de la placa de adaptación donde sea o no necesario una tensión de offset en el terminal de la placa adaptadora. Esto depende de si la señal a medir es siempre positiva o puede medirse como negativa.

- Si se pretende medir una señal que no pueda medir un valor negativo, como el caso de los sensores de tensión (V_d, V_o siempre positivos en el circuito), se puede trabajar sin Offset. Lo cual permite una mayor precisión, ya que los sensores tienen a su disposición los 16 bits del canal ADC, un valor del ADC total de 0 a 4095.
- Para una medida que pueda oscilar entre valores positivos y negativos como la intensidad de la bobina I_L, es necesaria la presencia de una tensión de Offset. Esta pone el valor cero de la medida del módulo al valor del ADC de 1930 aproximadamente, valor el cual depende de la calibración de los sensores. Esto permite tomar medidas positivas y negativas a costa de perder algo de precisión.

Módulos ADC	Pin.	Variable	Tensión de Offset
ADCIN_A2	64	V_o	No
ADCIN_B3	68	I_L	Si
ADCIN_C15	63	V_d	No

Tabla 5-3. Correspondencia de módulos ADC con pines del launchpad.

A la hora de la configuración hay que especificar los siguientes elementos:

- ADC Module: A, B, C.
- ADC Resolution: 12-bit (Single-ended input). La opción que permite medir una tensión por el pin.
- SOC trigger number: SOCx. Se escoge una para cada módulo. Hay que seguir una jerarquía en cuanto al módulo y SOCx, ya que estos se ejecutan con prioridad creciente desde el SOC0-SOC15.

Módulo ADC	A	B	C
SOCx	SOC0	SOC2	SOC4

Tabla 5-4. Asignación de Módulos ADCx con SOCx.

- SOCx acquisition window: El tiempo que va a estar el ADC operativo tomando datos. En este caso:

$$ADCCLK = SYSCLKOUT/5.0 = 200MHz/5.0 = 40\ MHz$$

$$Toma_{datosADC} = SOCx\ acquisition * ADCCLK = 10 * 40\ MHz = 400MHz = 400e^{-6}\ s$$

- SOCx trigger source: La fuente de disparo de los SOC. Como se ha habilitado el PWM7A para provocar una interrupción, se escoge 'ePWM7_ADCSOCA'.
- ADCINT Will trigger SOCx: No ADCINT. Ninguna interrupción de otro ADC va a disparar el SOCx.
- Sample time: -1 al ser llamado por la interrupción del PWM.
- Data type: uint16.
- Post Interrupt at EOC trigger: Interrupt selection ADCINT1. Esta opción se activa sólo en el ADCINA2. Cuando se alcance al final de conversión ('End of Conversion'), la interrupción ADCINT1 será disparada desde el módulo A.

  Como los tres módulos están ejecutándose al mismo tiempo, la interrupción habilitará el procesamiento de la información recopilada.
- Conversion channel: ADCINx. La entrada del ADC con el que se vaya a trabajar.

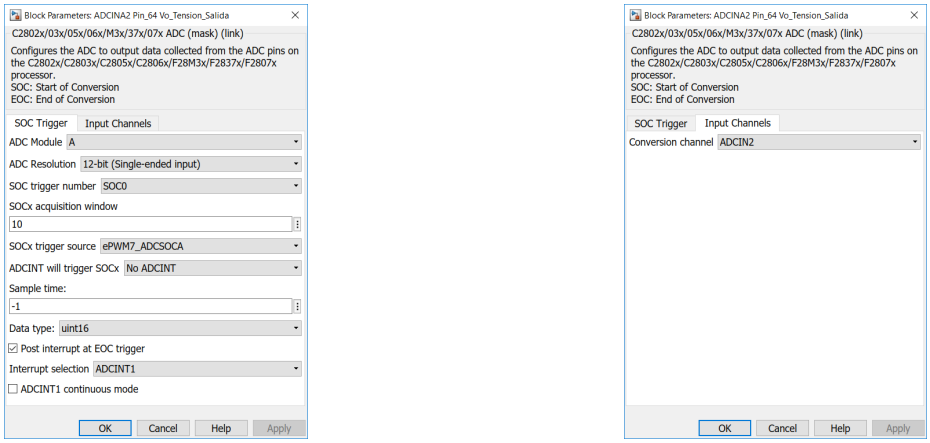


Figura 5-15. Diagrama Simulink Convertidor real. Área de Interrupción. Configuración ADC_A.

Nota: Los ADC_B, ADC_C, se programan de forma similar al ADC_A, con la configuración descrita previamente.

5.1.5. Tratamiento de las señales ADC.

Una vez los valores medidos de los ADC han sido adecuados al tiempo de ejecución, hay que convertir la los valores de señal obtenidos, al valor correspondiente a las variables medidas.

La entrada de cada ADC tiene un valor de $\{0 - 3{,}3\}\ V$ en el pin, que se traduce en un valor entre $\{0 - 4095\}$ a la salida del bloque de simulink. Con cada señal, se realizan dos operaciones en el diagrama.

- Mediante el bloque "Etapa de adaptación", se traduce el valor de la salida del ADC $\{0 - 4095\}$, al valor de la medida gracias al trabajo de calibración expuesto más adelante.
- Se va a crear un protocolo de transmisión, con el cual se estarán tomando 100 medidas de las magnitudes, y enviando durante el funcionamiento del concertidor. Estas magnitudes son V_o, V_d e I_L.

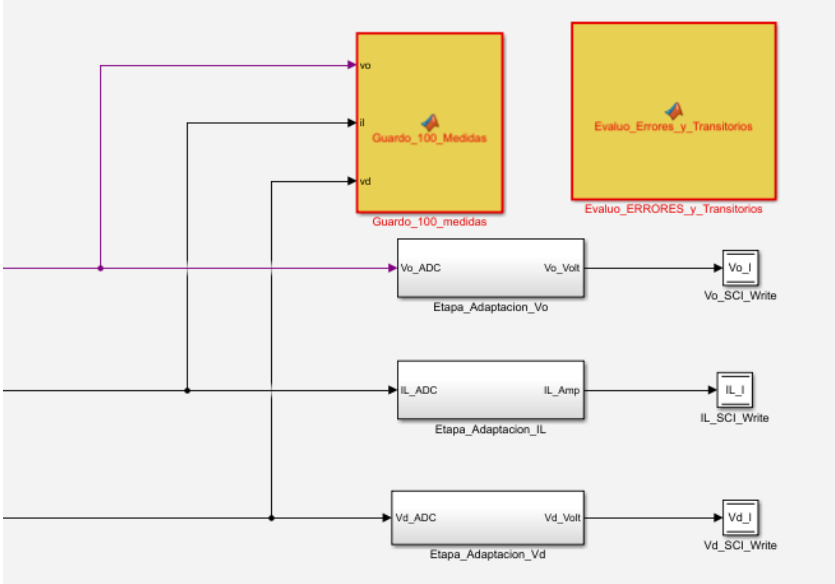


Figura 5-16. Diagrama Simulink Convertidor real. Área de Interrupción. Tratamiento señales ADC.

<u>Etapa de transmisión de los ADC vía comunicación serie.</u>

A la hora de enviar los valores de salida de los ADC, se trabaja con una comunicación via serie. En esta se enviarán los datos del bloque de simulink, que estará ejecutándose en el microcontrolador, de este al ordenador, donde se tratará la información.

La configuración de la comunicación serie vía SCI_A se va a desarrollar en su apartado.

Para enviar los datos de los ADC, hay que programar un protocolo de comunicación que trabaje como semáforo del sistema, cuando haya información que enviar y cuando no.

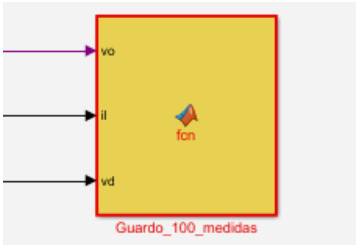


Figura 5-17. Diagrama Simulink Convertidor real. Área de Interrupción. Bloque función 100 medidas.

Este protocolo se implementa en la función descrita en los ANEXOS, “Guardo 100 medidas y las envio”.

En esta función se declaran una serie de variables como globales, ya que es necesario que su valor prevalezca a cada ejecución del programa. Funciona de la siguiente manera:

- Mientras la variable bandera “enable_flag” no esté activada, se guarda el valor medido por los ADC, en su respectivo buffer, correspondiente a las variables globales Vo_bus, Vd_bus e IL_bus. Estas variables globales son vectores de tamaño $(100,1)$, e inicialmente con valor cero.
- Cuando uno de los vectores esté lleno, es decir, que tenga 100 medidas guardadas, la variable global “enable_flag” actualiza su valor a 100. En este momento, se reinicia el índice i, al valor 1, para cuando vuelva a tomar medidas.
- Si es recibida por el puerto serie la orden de enviar información, la cual equivale a que la variable “s_mandar” valga 1, y además la variable “enable”, que habilita el envío por comunicación SCI también está a valor 1, se comienzan a enviar los valores guardados en la memoria, escribiéndolos en las variables globales, VoSCI, VdSCI e ILSCI.
- Cuando finalice la transmisión de los 100 datos, se reinicia el índice j y la bandera “enable_flag”.

Bloque función ‘Evaluo errores y transitorios’.

Este bloque se encarga de evaluar, si el convertidor se encuentra en el estado de ERROR, qué tipo de error está ocurriendo en el convertidor.

El código de este bloque se encuentra en los Anexos del proyecto.

Etapa de adaptación de ADC a la magnitud eléctrica.

Cada valor medido del ADC, debe ser tratado con una etapa de adaptación que la traduzca a la magnitud que está midiendo.

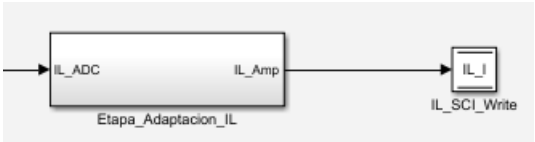


Figura 5-18. Diagrama Simulink Convertidor real. Área de Interrupción. Etapa adaptación ADC.

Esto se traduce en una ecuación de primer grado:

$$V_{medido} = K * ADC + V_{offset}$$

Donde,

V_{medido}, es el valor real de tensión o intensidad medido.

ADC, es el valor resultado a la salida del ADC $\{0 - 4095\}$.

K, es la ganancia, primera incógnita.

V_{offset}, es un valor de offset, segunda incógnita a calcular.

Para resolver esta ecuación se han usado dos ecuaciones definidas por Matlab, y que se pueden encontrar en los ANEXOS del proyecto, ‘Mido ADC puerto serie’ y ‘Calculo rectas de regresión’.

- Con la función 'Mido ADC puerto serie', se toman un total de 50 medidas de cada variable, para cada valor de tensión o intensidad medido. Después se calcula la media de esas 50 medidas.
- Con la función 'Calculo rectas de regresión', en primer lugar, se guardan los valores de media que son obtenidos de cada magnitud (V_o, V_d e I_L) en un vector, anotando el valor de la Tensión o Intensidad medido en otro.

  Una vez estén ambos vectores llenos, donde han sido recogidas un total de 11 mediciones para las tensiones y 10 para la intensidad, se calculan los valores de ganancia y offset de cada ADC.

La etapa de adaptación de cada ADC queda, una vez calculada:

Módulo ADC	V_o	V_d	I_L
Ganancia	0.053757109174382	0.052817860161452	0.005870195030470
Offset	-0.921479161736328	-1.248107976166655	-11.319073216832791

Tabla 5-5. Ganancia y Offset de cada salida ADC.

Traducido a bloques en Simulink de la forma:

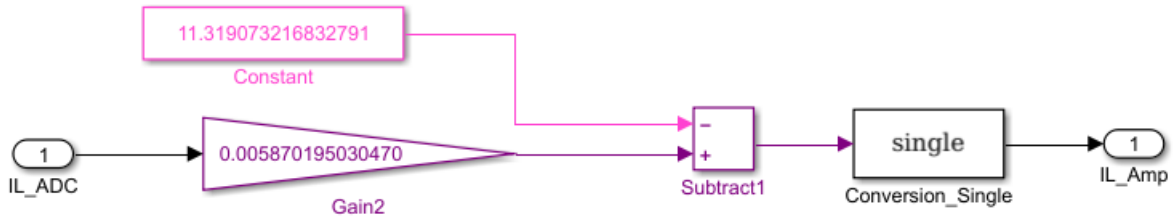


Figura 5-19. Diagrama Simulink Convertidor real. Área de Interrupción. Implementación adaptación ADC_IL.

5.1.6. Bloque de control PI del Convertidor.

El control del convertidor ahora tiene que realizarse cada vez que tomamos una medida, por lo cual se ha introducido dentro del bloque de la Interrupción del programa.

Las entradas son:

- Intendidad de la bobina I_L, medida por el ADC.
- Tensión de entrada V_d, medida por el ADC.
- Tensión de salida V_o, medida por el ADC.
- Señal de ENABLE, para habilitar funcionamiento de los PI.

La salida es el Duty Cycle calculado.

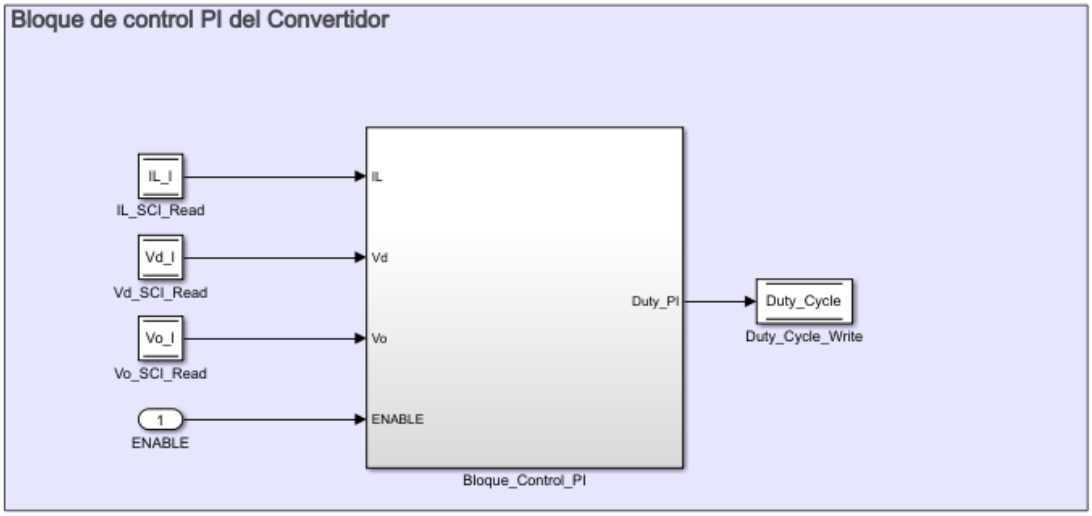


Figura 5-20. Diagrama Simulink Convertidor real. Bloque de control.

Tomado el trabajo ya realizado del apartado de la simulación virtual del convertidor, donde se ha diseñado los parámetros del PI de intensidad y tensión, para los valores requeridos.

Se puede observar que el valor de la Intensidad de referencia I_{ref}, a la salida del primer controlador PI, guarda su valor en memoria. Esta variable se usará para comprobar que el dispositivo funciona correctamente, y para representarla en la interfaz.

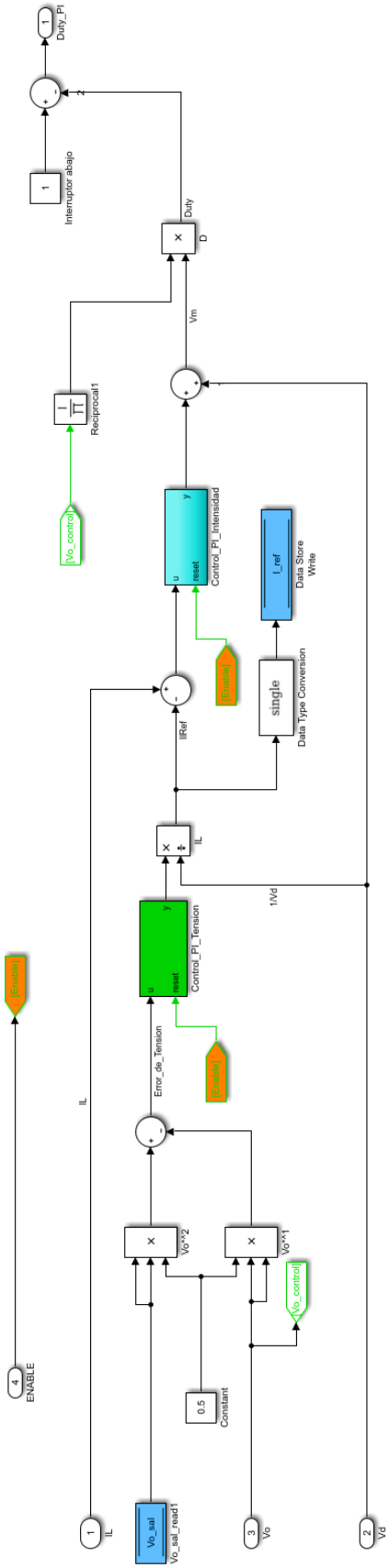


Figura 5-21. Diagrama Simulink Convertidor real. Bloque de control. Bloques PI.

Como los bloques PI se están ejecutando dentro de una interrupción, no pueden ser operados con la configuración estándar, así que se han construido los siguientes bloques, los cuales equivalen en funcionamiento a los diseñados en la simulación virtual.

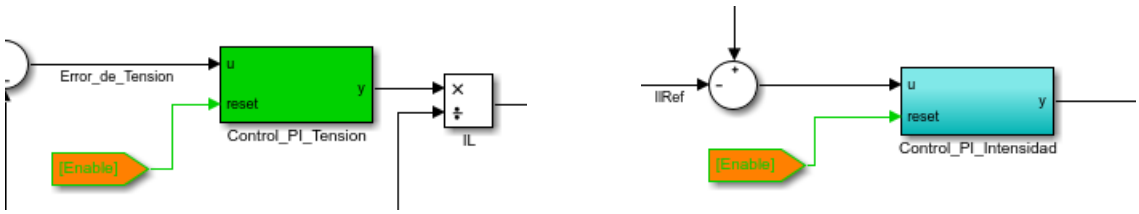


Figura 5-22. Diagrama Simulink Convertidor real. Bloque de control. Bloques diseñados PI.

Para el diseño de estos bloques, se ha reproducido el contenido de los bloques PI proporcionados por Matlab, donde se ha añadido de forma manual los valores de las ganancias calculadas.

En la configuración del bloque "Discrete-Time Integrator", se habilita, en la pestaña Initial condition setting, la opción output, con lo cual permite ejecutar los bloques PI dentro de la interrupción.

Como los componentes K_i y K_p de los convertidores van a a poder ser introducidos por la interfaz, deben ser expresadas como variables de memoria.

Al Bloque PI de tensión se le añade una saturación, para evitar grandes oscilaciones durante el funcionamiento.

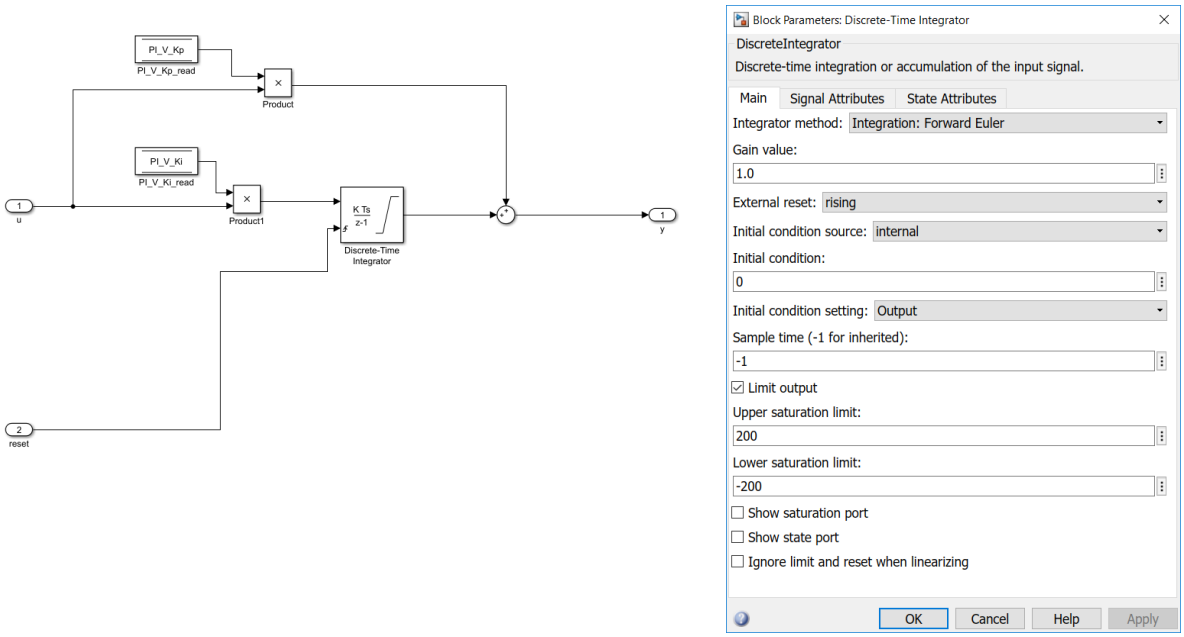


Figura 5-23. Diagrama Simulink Convertidor real. Bloque de control. Bloque PI diseñado.

Nota:

El duty cycle a la salida es calculado como el recíproco, restándole a 1 su valor. Esto es así ya que cuando la señal pasa al puente en H, está controlando al diodo contrario a los cálculos teóricos.

5.1.7. Etapas adaptadoras Sample Time.

Las señales que salen o entran del bloque de interrupción tienen que estar definidas correctamente en cuanto al tiempo de ejecución, ya que algunas se ejecutarán de forma asíncrona, y otras a un tiempo definido.

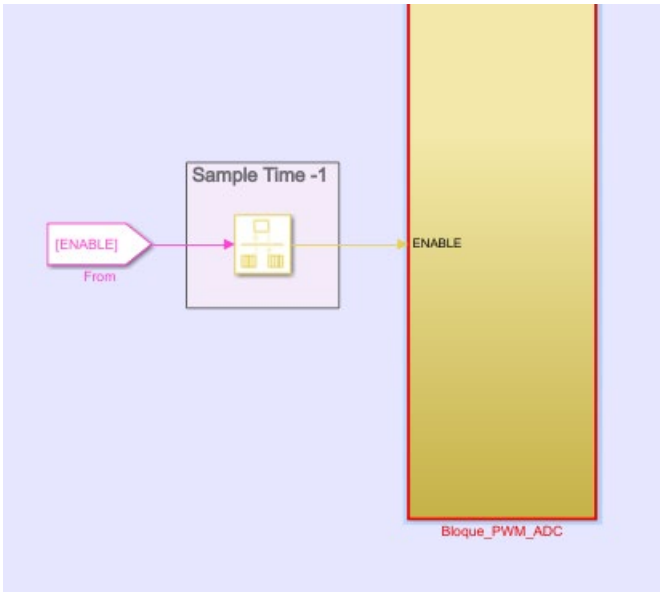


Figura 5-24. Diagrama Simulink Convertidor real. Área de Interrupción. Etapas adaptadoras Sample Time.

Las entradas al bloque de interrupción deben tener un Sample Time de valor -1. Lo cual permite que sean ejecutadas de forma asíncrona, que es lo que necesitan al estar ejecutándose el bloque por la interrupción del ADC.

En el bloque de interrupción, se trabaja con variables globales, las cuales estarán actualizadas a lo largo de la operación del programa. No es necesario tratar dichas variables con un sample time, ya que no hay un tiempo determinado fuera del bloque de la interrupción. Cuando una variable le llega un nuevo valor, se actualiza.

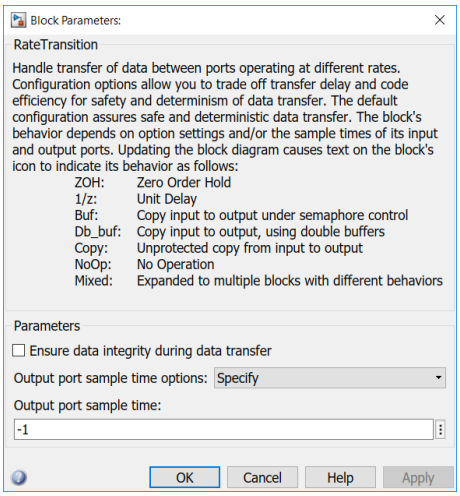


Figura 5-25. Diagrama Simulink Convertidor real. Área de Interrupción. Sample Time asíncrono.

5.2. Programación de GPIO Digital Input.

El circuito de Puente en H, puede avisar del mal funcionamiento del mismo, es decir, que envía un error cuando no está operando como es debido.

Este error se envía mediante el terminal de Transmisión Tx del Puente en H, y se recoge por el segundo terminal de Recepción Rx de la etapa adaptadora de Fibra óptica.

Este error se tratará en la máquina de estados, donde se interrumpirá cualquier funcionamiento si está activado. Por la inversión de los terminales Rx, hay que tener en cuenta que cuando no haya error, el pin está a 3,3 V, y el valor de GPIO está a 1.

Con un bloque de Negación a la salida, se opera con normalidad, cuando no esté el puente en H conectado.

Para la transmisión de datos vía comunicación serie, se convertierte la señal al tipo $uint16$ antes de guardar su valor en la variable global.

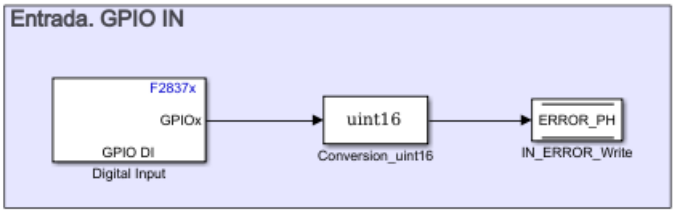


Figura 5-26. Diagrama Simulink Convertidor real. Área de Interrupción. Error Puente en H – Fibra óptica.

Este terminal de Rx del error en el puente en H está conectado a la GPIO 41, que corresponde al pin 5 del microcontrolador.

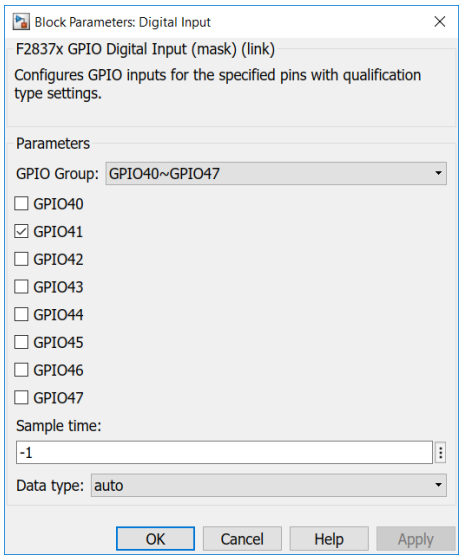


Figura 5-27. Diagrama Simulink Convertidor real. Área de Interrupción. Configuración GPIO_IN.

5.3. Área Comunicación Serie.

El diagrama de Simulink programado, se ejecutará en el microcontrolador, una vez se instale con la opción "Deploy to Hardware" de simulink. Es necesario establecer algún tipo de comunicación entre el microcontrolador y la computadora.

En el programa se quiere recibir y transmitir información, de forma que son habilitados los bloques:

- SCI_Transmit.
- SCI_Receive.

El bloque de recepción, como se pretende tratar las señales desde el momento que lleguen, toma las señales y las asigna tal cual llegan.

En cambio, para la transmisión hay que programar un protocolo de comunicaciones. Para esto, se ha diseñado una máquina de estados, y un bloque con señal "enable", para que cuando se cumpla la restricción de comunicación se proceda a enviar los datos.

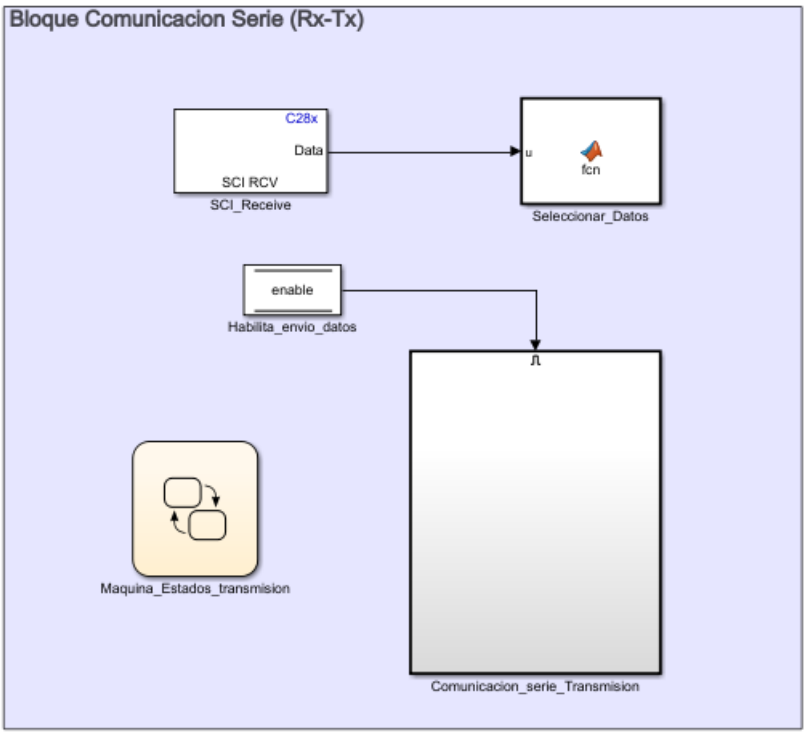


Figura 5-28. Diagrama Simulink Convertidor real. Bloque de comunicación serie.

Para este desempeño, se ha escogido la comunicación serie SCI_A. Los pines asignados de este módulo de comunicación, corresponden al cable USB.

SCI_A	Tx	Rx
Pin	GPIO84	GPIO85

Tabla 5-6. Asignación de GPIO de comunicación SCI_A.

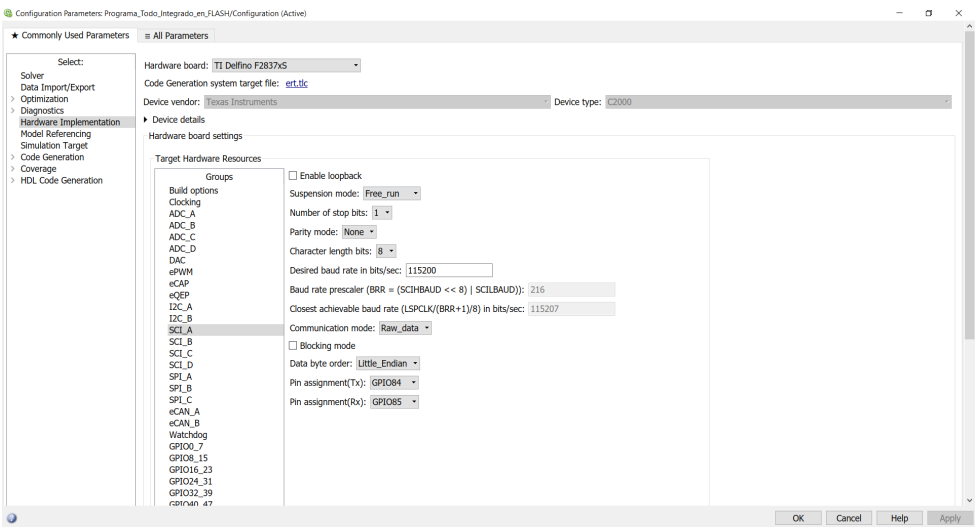


Figura 5-29. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Configuración SCI_A.

5.3.1. Recepción de la información.

El bloque 'SCI-Receive' trata toda la información que llegue al microcontrolador por via SCI_A.

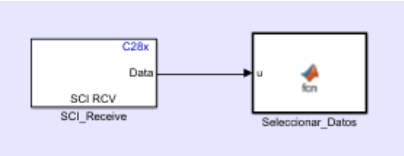


Figura 5-30. Diagrama Simulink Convertidor real. Bloque de comunicación serie. SCI_Receive.

Se configura con los siguientes parámetros.

- SCI module: A. El módulo donde se programa la comunicación serie.
- Additional package header/terminator: ''.
- Data type: uint16. Formato en el que se vana a enviar los mensajes.
- Data lenght: 10. Las órdenes vendrán dadas por un vector de tamaño diez, donde se recibe:
    - "ESTADOS", variable que indica el paso de un estado a otro.
    - "s_mandar", señal que indica el comienzo de la transmisión de las medidas de los ADC, si hubiera información que enviar.
    - "Vo_sal", valor de la tensión deseada a la salida.
    - "PI_V_Kp, PI_V, Ki, PI_I_Kp y PI_I_Ki", valores de los parámetros de los bloques PI.
- "Limite_Vo, IL, Vd", valores máximos permitidos del circuito. Initial output: 0.
- Action taken when connecction times out: Output the last received value.
- Sample time: -1.

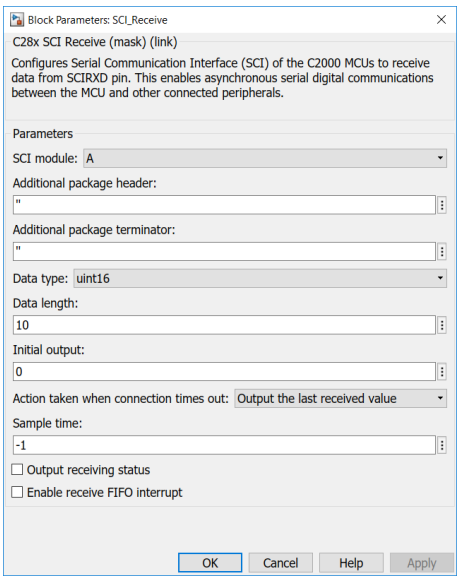


Figura 5-31. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Configuración SCI_Receive.

A la salida del bloque de recepción, se ha situado una función de Matlab, 'Seleccionar Datos', que se encargará de filtrar la información que llegue por el bloque de recepción, otorgando a las variables globales, los valores que le corresponden del vector de llegada.

Como los datos recibidos pueden tener valores decimales, se va a enviar la información multiplicada por 32 o 100, dependiendo de la exactitud que se pretenda tener.

Variable Global.	Valor del vector de recepción.	Tratamiento de la variable.
$ESTADOS$	$u(1)$	–
s_mandar	$u(2)$	–
Vo_sal	$u(3)$	32
PI_V_Kp	$u(4)$	100
PI_V_Ki	$u(5)$	32
PI_I_Kp	$u(6)$	32
PI_I_Ki	$u(7)$	32
$Limite\_Vo$	$u(8)$	32
$Limite\_IL$	$u(9)$	32
$Limite\_Vd$	$u(10)$	32

Tabla 5-7. Asignación de datos llegada por bloque de recepción.

5.3.2. Protocolo deTransmisión por SCI_A.

A la hora de transmitir los datos del micro al ordenador, se ha diseñado el siguiente protocolo:

- Del bloque interrupción, se guardan 100 medidas de cada canal ADC, activándose la señal bandera.
- Si la señal bandera está activa, y es recibida la orden de comenzar transmisió, se envían las 100 medidas.

Esto se llevará a cabo mediante una máquina de estados, que cuando se dan las condiciones pertientes active un enable al bloque de transmisión.

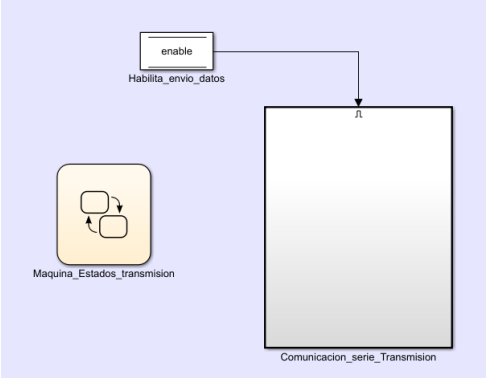


Figura 5-32. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Protocolo Transmisión.

5.3.3. Transmisión de los datos.

Señal habilitante del bloque de transmisión.

La información debe enviarse cuando haya sido tratada, no se pretende obtener un torrente sin orden de información. Se introduce SCI_Transmit en un bloque, que gracias a "Enable" de Matlab, será habilitado con una señal.

Para habilitarlo, se crea la variable global $enable$, tratada desde la máquina de estados de transmisión.

Bloque de transmisión de información.

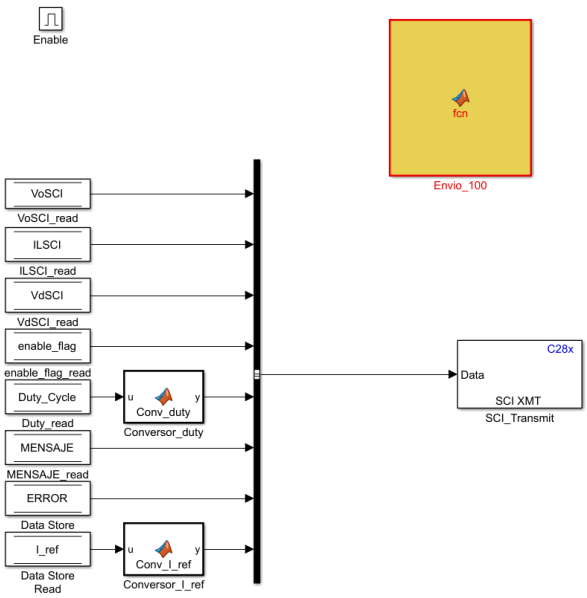


Figura 5-33. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Transmisión.

La variable “Enable” es configurada de la siguiente forma, para que habilite la funcionalidad del bloque.

- States when enabling: held.
- Propagate sises of variable-size signals: Only when enabling.

Con esto, cuando la señal que llegue valga 0, el bloque estará desactivado. Cuando llegue un 1, se habilitará la transmisión de datos.

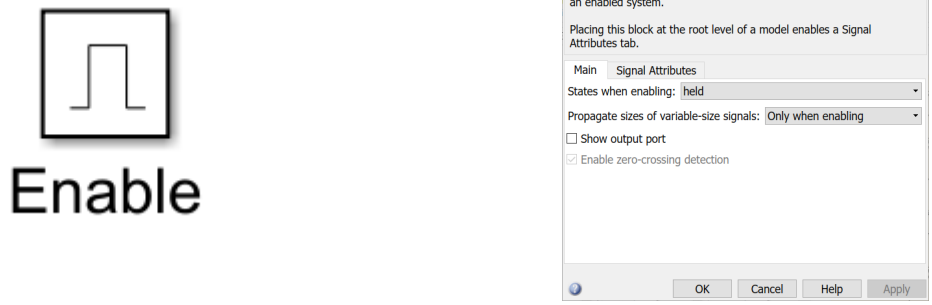


Figura 5-34. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Configuración Enable.

La información a enviar del microcontrolador al ordenador es:

- Valor V_o medido por ADC.
- Valor I_L medido por ADC.
- Valor V_d medido por ADC.
- Variable global enable_flag, de protocolo de comunicación (función Guardo 100 medidas y envío).
- Valor $Duty\_Cycle$, calculado a la salida del sistema de control.
- Variable global MENSAJE, correspondiente al estado de ejecución de la máquina de estados.
- Variable global ERROR, que indica si hay algún error en el convertidor.
- Valor I_ref, calculado dentro del sistema de control.

Estas señales mediante el bloque de simulink “bus creator”, se unen para la transmisión de la información.

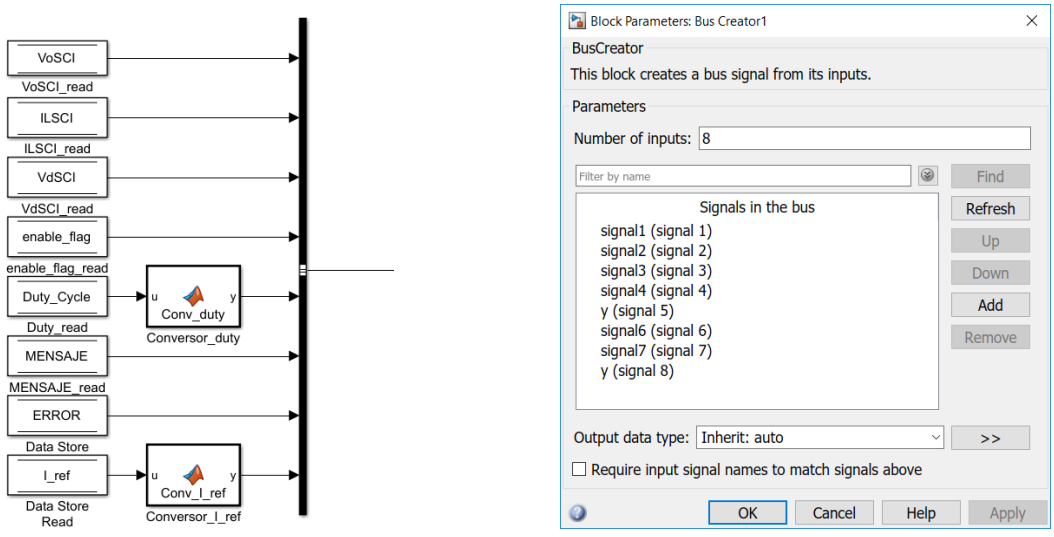


Figura 5-35. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Señales bus.

Las funciones "Conversor_duty" y "Conversor_I_ref", son funciones de Matlab accesibles en los Anexos del proyecto. Su función es multiplicar la variable $Duty\_Cycle$ e I_ref por 1000 y por 100 respectivamente.

Como ambas son decimales, no superan el valor 1 (la intensidad puede, pero no en los limites quese va a probar el elevador)., siendo el límite de precisión de 16 bits 4096, podrán ser enviados estos valores decimales.

Será necesario tratar en el programa de recepción (la interfaz gráfica) estos valores, dividiéndolos por 1000 y por 100 antes de representar, u operar con ellos.

El bus de datos, llega al bloque de SCI_Transmit, que es configurado:

- SCI module: A.
- Additional package header: 'A'.
- Additional package terminator: 'B'.

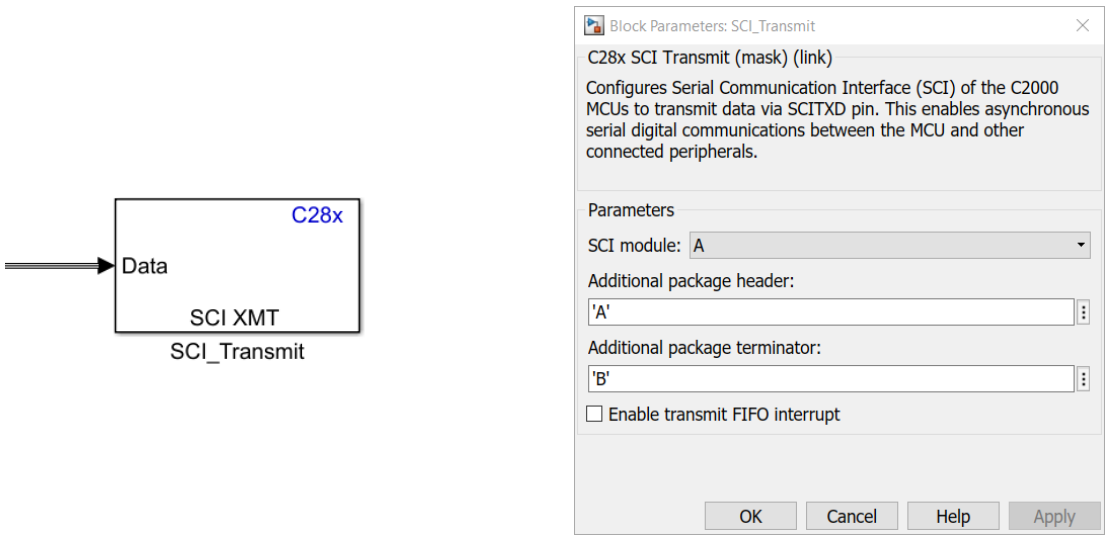


Figura 5-36. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Configuración SCI_Transmit.

Datos transmitidos por comunicación SCI.									
'A'	V_oADC	I_LADC	V_dADC	$enable\_flag$	$Duty\_Cycle$	$MENSAJE$	$ERROR$	I_ref	'B'

Tabla 5-8. Datos transmitidos por comunicación SCI.

Función envio 100.

Esta función de Matlab se encarga de que, una vez preparado el programa para enviar información, los datos guardados en los buses de memoria (tensiones e intensidad, medidos por los ADC) se copien en las variables $VoSCI, ILSCI$ y $VdSCI$, para que se envíen por el puerto serie.

5.3.4. Máquina de estados de transmisión de datos.

Finalmente, se va a configurar la máquina de estados, con la que se podrá implementar el protocolo de transmisión de información que se ha diseñado. Sigue la estructura:

- Se parte del estado INICIO, inicializador, del cual pasamos a TOMANDO_MEDIDAS.
- En TOMANDO_MEDIDAS, la variable que habilita la transmisión, $enable$ adquiere el valor 0. Se mantiene en el estado mientras $i < 101$.
- En el momento que llega la bandera $enable\_flag$, cambia al estado SENAL_MANDAR.
- En este estado puede ocurrir dos opciones.
    - Si la señal s_mandar no se ha recibido (tiene valor 0), se vuelve al estado previo, TOMANDO_MEDIDAS, y se reinicia la toma de datos. Con esto se consigue decartar medidas previas que no poseen relevancia.
    - Si la señal s_mandar ha llegado y tienen valor 1, se pasa al estado ENVIO_MEDIDAS.
- En el estado ENVIO_MEDIDAS la variable $enable$ adquiere el valor de 1, con lo que se habilita el bloque de transmisión de la información. No cambia el estado mientras la variable $j < 101$.
- En el momento que la señal bandera se vuelve $enable\_flag = 0$, significa que se han mandado las 100 medidas tomadas. Se reinicia la máquina de estados, volviendo a TOMANDO_MEDIDAS.

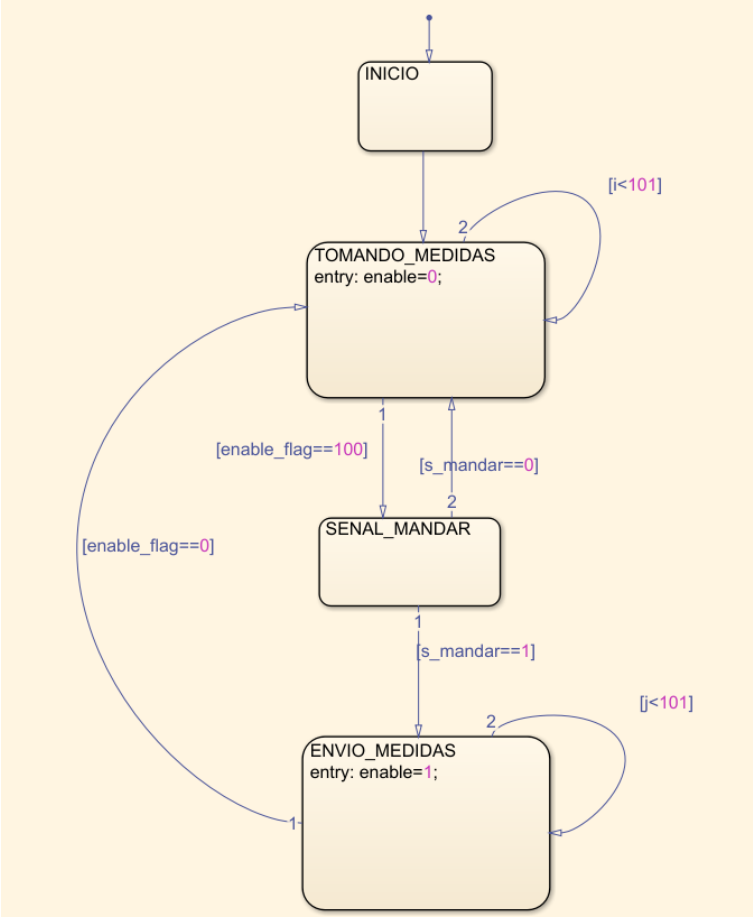


Figura 5-37. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Máquina de estados transmisión.

5.3.5. Prueba de recepción de información.

Para comprobala tranmisión de datos del microcontrolador al ordenandor via comunicación serie se va a realizar una prueba de toma de medidas, mediante el uso de uno de estos dos programas se puede visualizar qué está enviando el microcontrolador. Se implementa el diagrama de simulink mediante “Deploy to hardware”.

Nota: Estas pruebas se realizaron con una versión anterior del programa durante su desarrollo, por lo que no están reflejadas todas las variables del programa final.

Prueba de recepción con Realterm.

El programa empleado será “Realterm”, fácil de instalar e intuitivo en cuanto a su funcionamiento.

En Port, se configura qué puerto COMx está conectado al micro, con las mismas características implementadas.

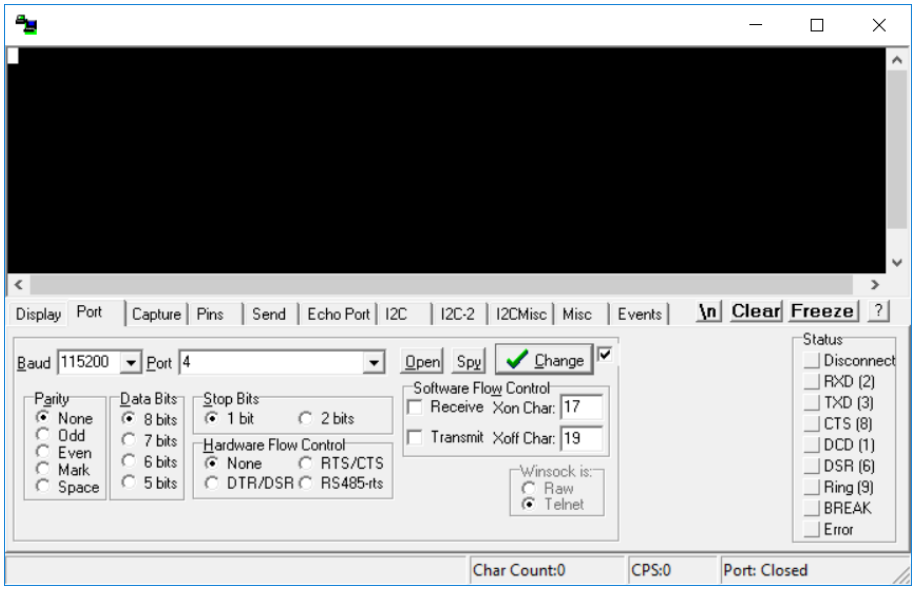


Figura 5-38. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Prueba Realterm Port.

Se abre el puerto, y en Display se escoge la forma de representación que se desea de la información.

Ejecutando como ASCI, se observan los caracteres ‘A’ y ‘B’ entre el código enviado.

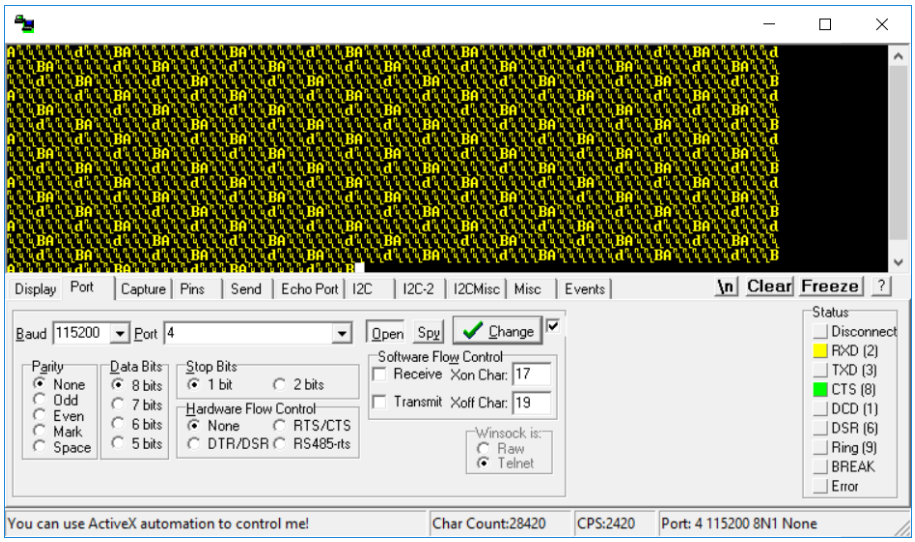


Figura 5-39. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Prueba Realterm ASCII.

Puesto es necesaria una señal para que los valores de los ADC se envíen, no se va a observar más que un valor cero. Sin embargo, el contador de carga del buffer está llenándose, y una vez lleno actualiza su valor a 100.

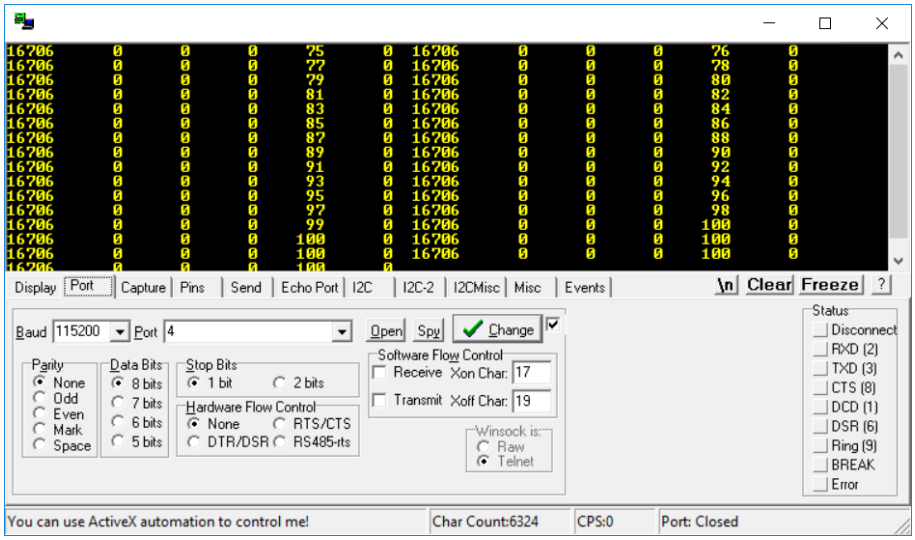


Figura 5-40. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Prueba Realterm, uint16.

Prueba de recepción con Matlab.

También se puede ver la transmisión del micro con Matlab, para lo cual se ha generado la función descrita en los Anexos Probando_Recepcion. Este programa, creará una conexión serie con el puerto 'COMx', y tomará una serie de medidas de los datos que reciba.

Explicando el programa:

- Se elimina cualquier uso que se le esté dando a los puertos serie.
- Tras esto, se crea una conexión de comunicación serie en el puerto 'COMx', a la velocidad 115200 bauds. Se abre la conexión creada.
- Puesto que el protocolo de comunicación necesitaba de una orden de recepción, se envía con la función fwrite, la orden de envío de datos.
- Hay que crear una variable "entrada_adc" donde al ejecutar la función fread, se guardan los primeros 100 datos recibidos en formato uint16.
- A continuación, se convierte este vector de datos uint16, en uno de tipo char. Puesto que tienen la mitad de tamaño (8 bits), se genera el doble de información.
- Por último, hay que cerrar la conexión con el puerto serie, y eliminar como en el primer paso.

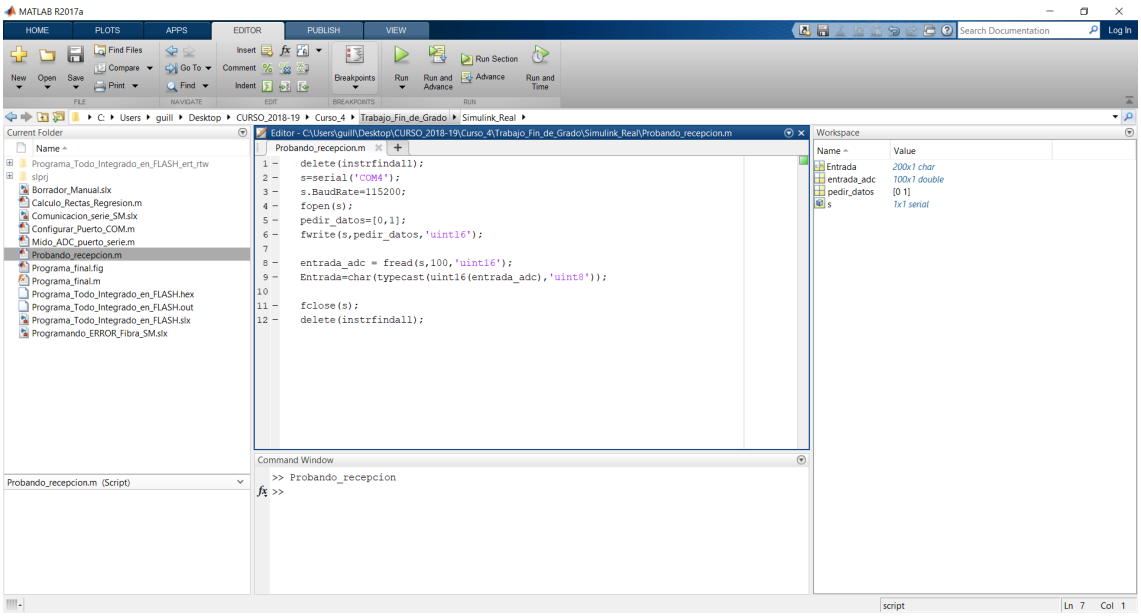


Figura 5-41. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Prueba Matlab.

En el vector de datos “entrada_adc”, se obtienen las medidas que esperadas:

- Valor correspondiente a los caracteres de comienzo, fin de tranmisión.
- Valor del $ADCV_o$, cuyo pin está conectado a 3,3 $V \rightarrow 4095\ ADC$.
- Valor $ADCI_L$, cuyo pin no está conectado.
- Valor $ADCV_d$, cuyo pin no está conectado.
- Bandera $enable\_flag$ de 100 medidas guardadas.
- Variable $MENSAJE$, correspondiente al estado 0.

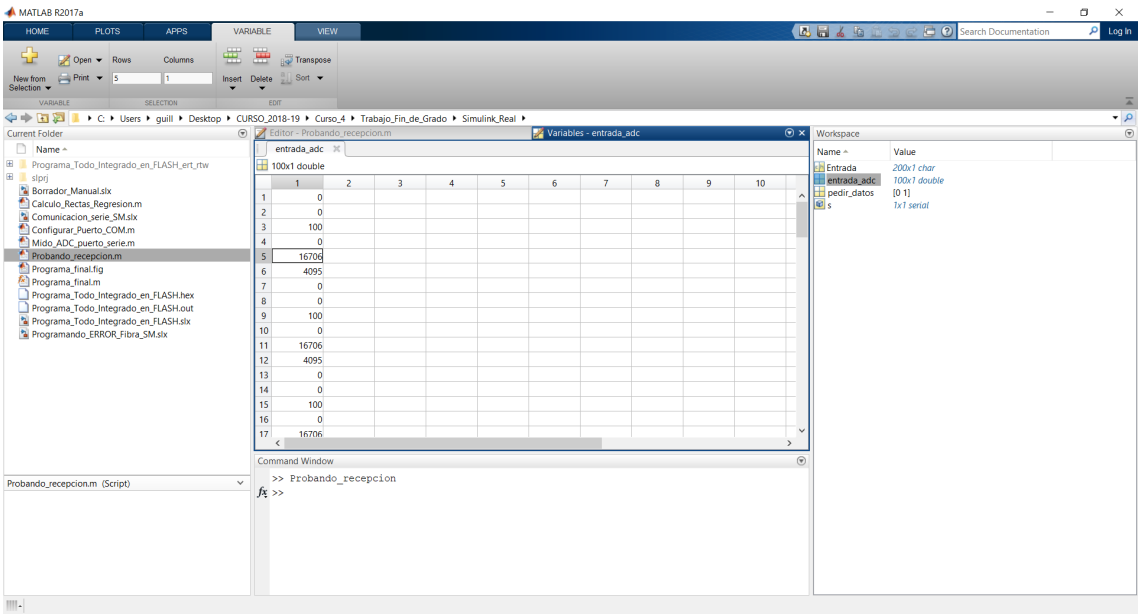


Figura 5-42. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Resultado Matlab.

En el vector “Entrada”, se convierte a tipo char toda la información procedente de “entrada_adc”. Este vector es útil, ya que permite visualizar los caracteres de inicio, final de transmisión $'A', 'B'$.

Gracias a estos caracteres, es posible situar un punto de partida, a partir del cual comenzar el tratamiento de la información recibida por el puerto serie.

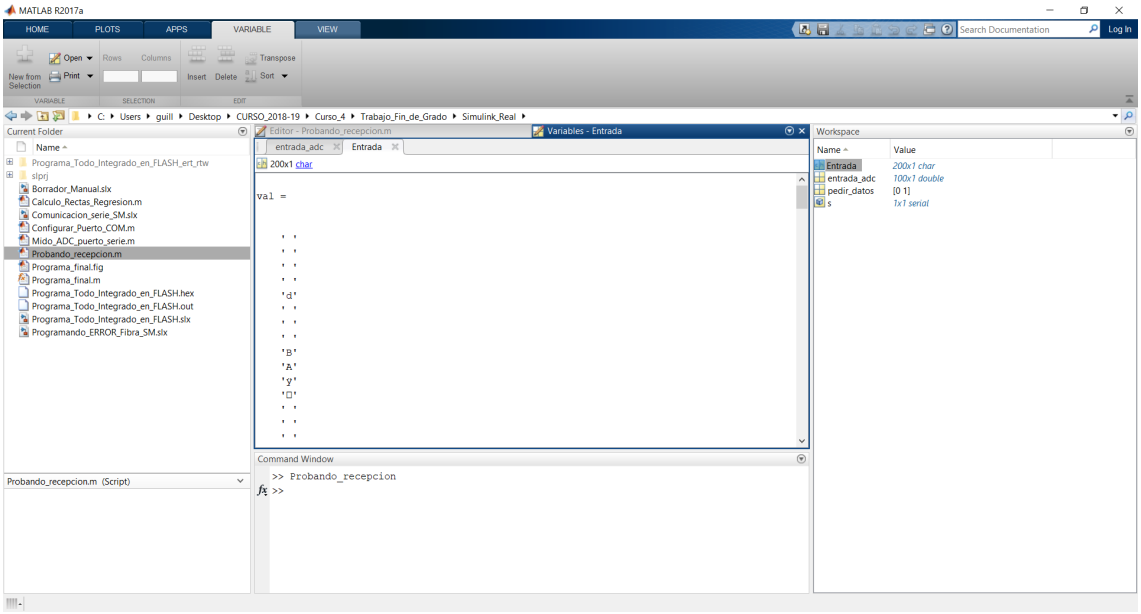


Figura 5-43. Diagrama Simulink Convertidor real. Bloque de comunicación serie. Resultado Matlab, Entrada.

5.4. Máquina de estados.

En el convertidor real, vamos a tener un modo de operación más complejo que en la simulación virtual. Por esto, se va a rediseñar la máquina de estados que controle los distintos modos de funcionamiento del convertidor.

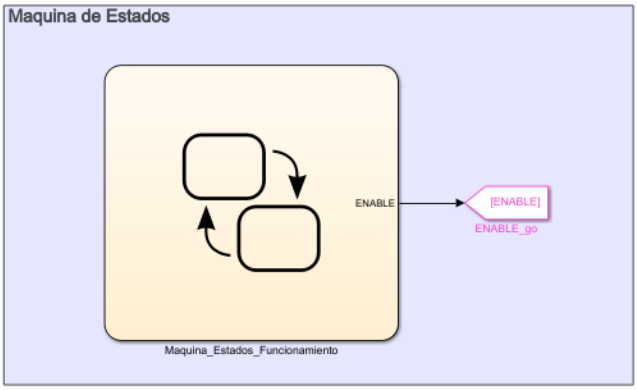


Figura 5-44. Diagrama Simulink Convertidor real. Máquina de estados.

La máquina de estados va a trabajar con las siguientes variables globales:

- ESTADOS. Es la variable de transición de la máquina de estados. Cuando se quiera cambiar de un estado a otro, es enviada por la comunicación serie una orden.

  Esta orden será un valor entero, del cual dependerá el paso entre estados.

- ERROR_PH. Variable que indica un error en el elemento electrónico puente en H.

  Si la máquina de estados se encuentra en algún estado de operación y de repente el circuito deja de funcionar, enviará una señal de error, con lo que la máquina pasará a un estado donde se aborta el funcionamiento del convertidor

- ARRANQUE. Es la variable que permite el funcionamiento de los módulos A y B del PWM7.
- ERROR. Actua como bandera, del error. Una vez se recargue el elevador que ha tenido un error, se desactiva la bandera, y se volverá al estado de APAGADO.
- MENSAJE. Se envía en el bloque de SCI_Transmit. Gracias a este valor, se podrá saber en qué estado se está ejecutando el programa desde el ordenador.

La salida de la máquina de estados corresponde a:

- ENABLE. Variable que habilita el bloque de control, reseteando los bloques PI, para eliminar valores anteriores al control, para conseguir alcanzar el valor deseado de V_o.

ESTADO	APAGADO	PRECARGA	OPERANDO	DESCARGA	ERRORES	RECARGA
ARRANQUE	0	1	1	1	0	0
ENABLE	0	1	1	0	0	0
MENSAJE	0	1	2	3	4	0
ERROR	-	-	-	-	-	0

Tabla 5-8. Estados y valor de variables de salida de la máquina de estados.

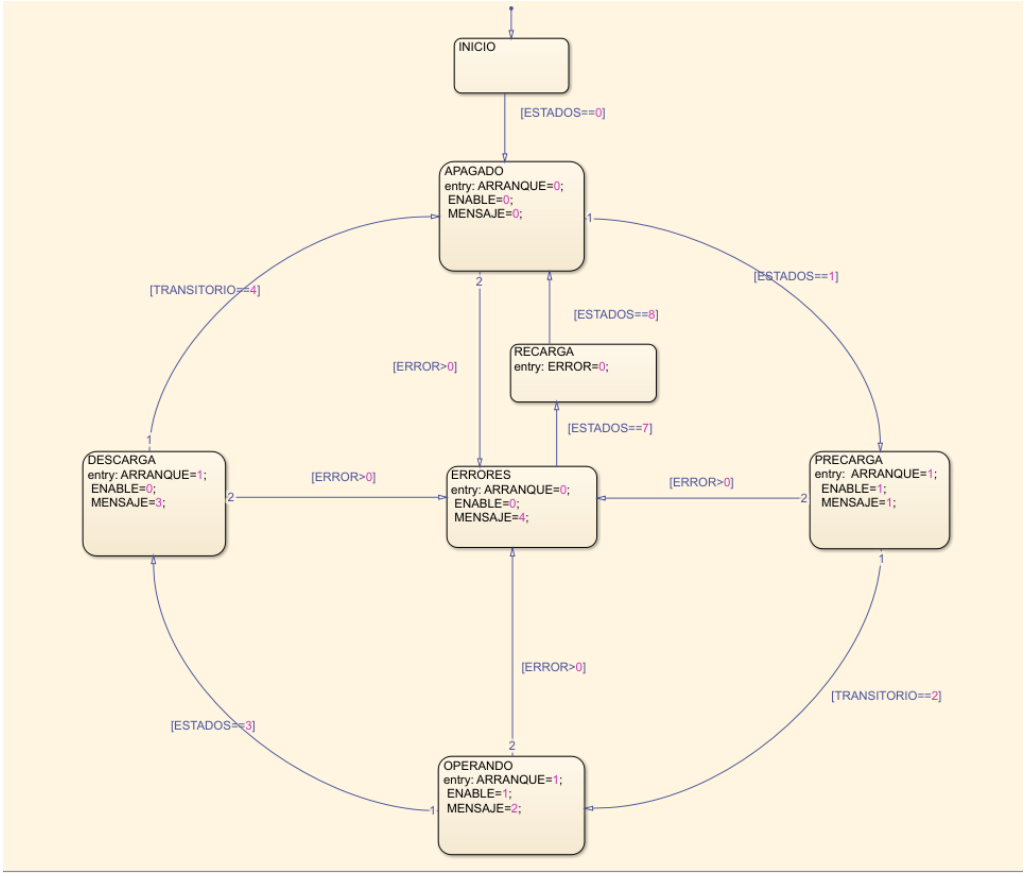


Figura 5-45. Diagrama Simulink Convertidor real. Máquina de estados. Bloques y transiciones.

La máquina de estados opera de la siguiente manera:

- Comienza en el estado INICIO. Este actúa únicamente como punto de partida al iniciar el programa. Pasará a APAGADO automáticamente.
- Si se pulsa el botón de la interfaz de control 'Operación', se actualiza el valor de la variable ESTADOS a 2. Esto hace cumplir la transición, pasando al estado PRECARGA.

  En este estado se habilita el módulo PWM $\{ARRANQUE = 1\}$ y además la señal ENABLE adquiere valor de 1, que resetea los bloques de control PI, activando a su vez el control.
- Para completar el transitorio, en la función "Evaluo Errores y transitorios", la tensión medida a la salida debe valer el 75% de la tensión de salida esperada. Tras esto se pasa al estado OPERANDO.
- Si se pulsa el botón de la interfaz de control 'Desconectar', se actualiza el valor de la variable ESTADOS a 3. Esto hace cumplir la transición, pasando al estado DESCARGA.

  En este estado se desactiva el módulo de control $\{ENABLE = 0\}$, por lo que la tensión comienza a descender.
- Para completar el transitorio, en la función "Evaluo Errores y transitorios", la tensión de salida debe menor a la tensión de entrada. Si se cumple, se pasa al estado APAGADO.
- En cualquiera de los estados, puede currir un error, que el puente en H falle, o que las tensiones o la intensidad sean mayores que el límite impuesto. Esto hace que la máquina pase a un estado de ERRORES, además se evalua el tipo de error en la función "Evaluo Errores y transitorios".

  Para salir del estado de ERRORES, hay que pulsar los actuadores de la interfaz de control: 'Comprobar error', lo que nos lleva al estado de RECARGA, y 'Recarga', que reinicia el sistema en el estado APAGADO.

5.5. Variables globales. Reserva de memoria.

La ejecución del diagrama de Simulink se realiza dentro del microcontolador, donde se ha introducido el programa con la opción "Deploy to hardware".

En cada periodo de ejecución del programa, el micro reescribe los valores de sus variables cada vez que se ejecuta. Esto supone un problema ya que, al no guardarse los valores de estas variables, se pierden.

Puesto que en el programa hay una serie de funciones de Matlab, principalmente para el tratamiento de la información, es necesario crear una serie de variables globales, que además mantengan su valor y se vayan actualizando a medida que el programa se ejecute.

Esto se consigue trabajando con los bloques de reserva de memoria "Data Store Memory", con los cuales es posible crear una variable global, que disponga de un área de la memoria del microcontrolador, donde poder almacenar su valor, e irse actualizando.

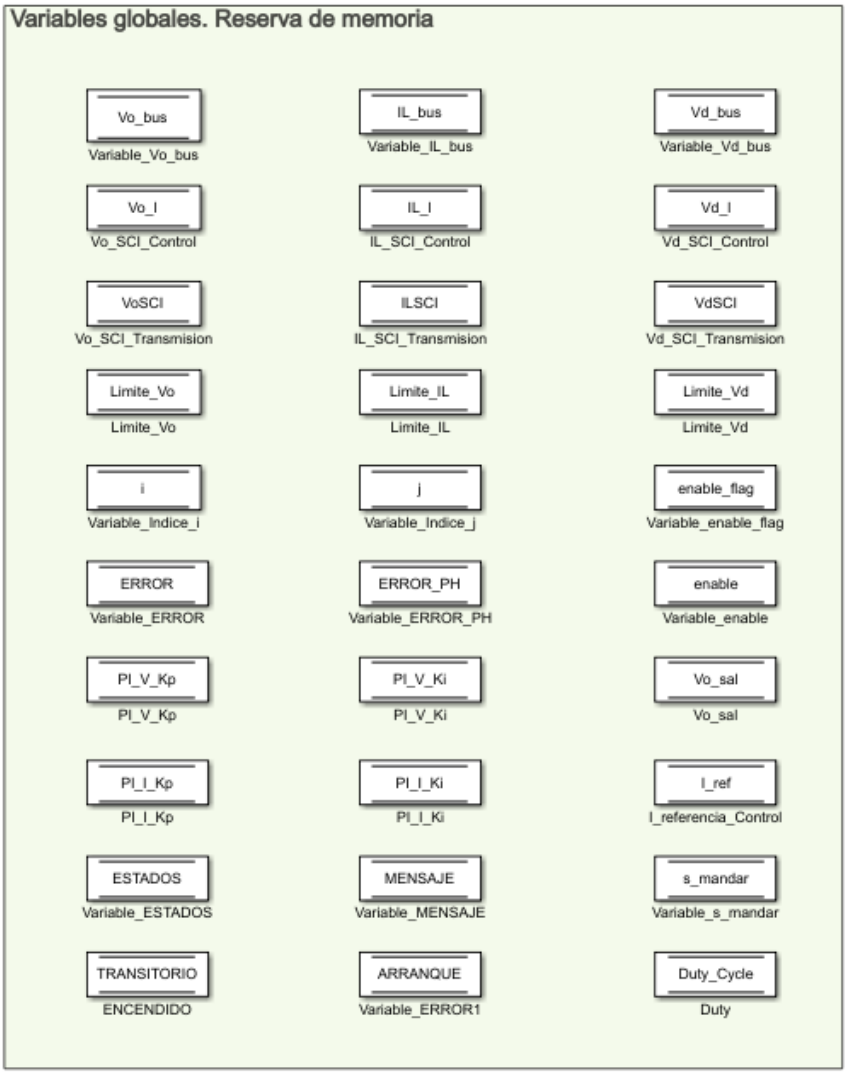


Figura 5-46. Diagrama Simulink Convertidor real. Área de reserva de memoria.

En los ajustes de parámetros de los “Data Store Memoy” se puede acceder a la información de las salidas (lectura) o entradas (escritura) de la variable en el programa.

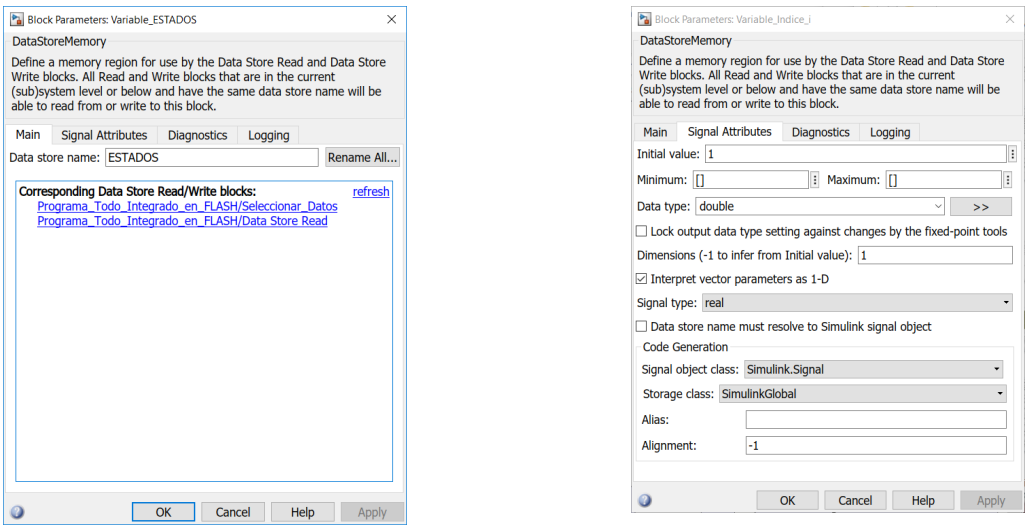


Figura 5-47. Diagrama Simulink Convertidor real. Área de reserva de memoria. Muestra de lectura-escritura (1). Signal atributes ejemplo (2)

Hay que configurar el apartado “Signal Attributes” en Code Generation:

- Signal type: real.
- Signal object class: Simulink.Signal.
- Storage class: SimulinkGlobal.

El tamaño, valor inicial y tipo de dato depende de cada variable.

Variable.	Valor inicial	Tipo.	Tamaño.
Vo_bus	0	$uint16$	100
IL_bus	0	$uint16$	100
Vd_bus	0	$uint16$	100
Vo_I	0	$single$	1
IL_I	0	$single$	1
Vd_I	0	$single$	1
$VoSCI$	0	$uint16$	1
$ILSCI$	0	$uint16$	1
$VdSCI$	0	$uint16$	1
$Limite\_Vo$	100	$single$	1
$Limite\_IL$	5	$single$	1

$Limite\_Vd$	60	$single$	1
i	1	$uint16$	1
j	1	$uint16$	1
$enable\_flag$	0	$uint16$	1
$ERROR$	0	$uint16$	1
$ERROR\_PH$	0	$uint16$	1
$enable$	0	$uint16$	1
PI_V_Kp	0.01	$single$	1
PI_V_Ki	0.1	$single$	1
Vo_sal	100	$single$	1
PI_I_Kp	3	$single$	1
PI_I_Ki	100	$single$	1
I_ref	0	$single$	1
$ESTADOS$	0	$uint16$	1
$MENSAJE$	0	$uint16$	1
s_mandar	0	$uint16$	1
$TRANSITORIO$	0	$uint16$	1
$ARRANQUE$	0	$uint16$	1
$Duty\_Cycle$	0	$double$	1

Tabla 5-9. Programación de las variables de memoria.

Formas de acceder a las variables de memoria:

1. Si se está tratando la señal como bloques de simulink, a la hora de acceder a la información de las variables para ver su valor o actualizarlo, se emplean los bloques:

- Data Store Read.
- Data Store Write.

Figura 5-48. Diagrama Simulink Convertidor real. Área de memoria. Bloques DataStoreRead/Write

2. En la máquina de estados se están actualizando las variables:

- Se abre la ventana “Explore”.
- Se selecciona la variable ESTADOS.
- En el apartado “Scope”, es escogido el tipo de la variable como “Data Store Memory”.

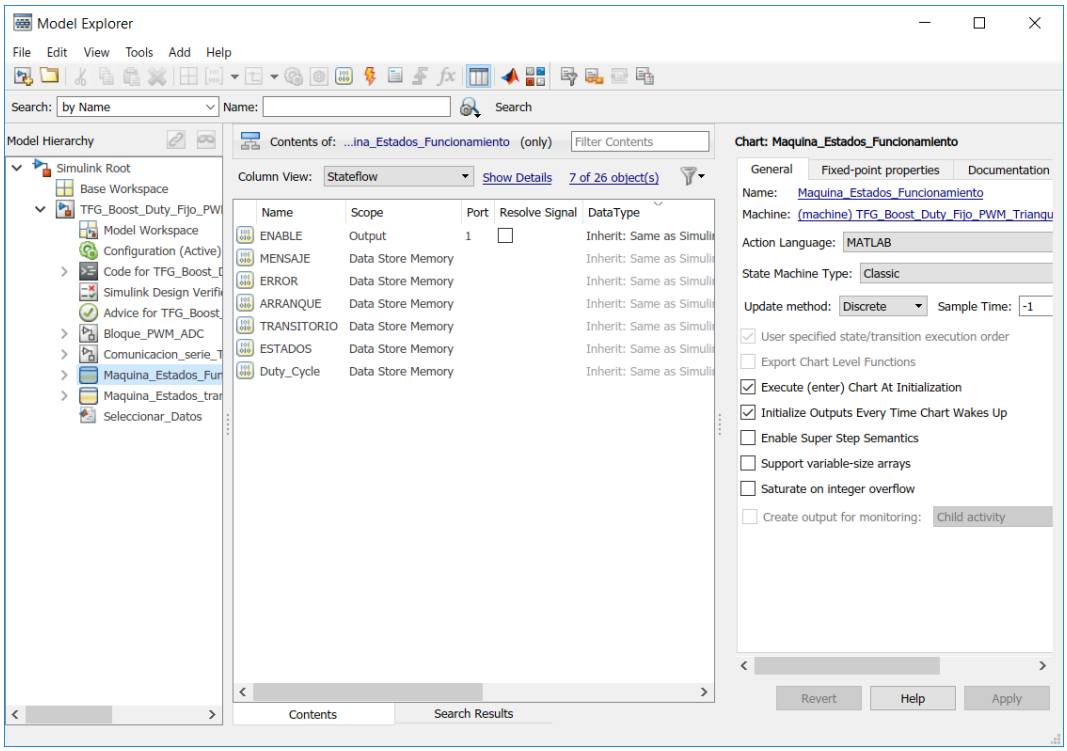


Figura 5-49. Diagrama Simulink Convertidor real. Área de memoria. Variable de máquina de estados.

3. Dentro de un bloque de función de Matlab hay que seguir los siguientes pasos.

   Se abre la función que accede a las variables globales. En “Editor” de Matlab se ejecuta “Edit Data”.

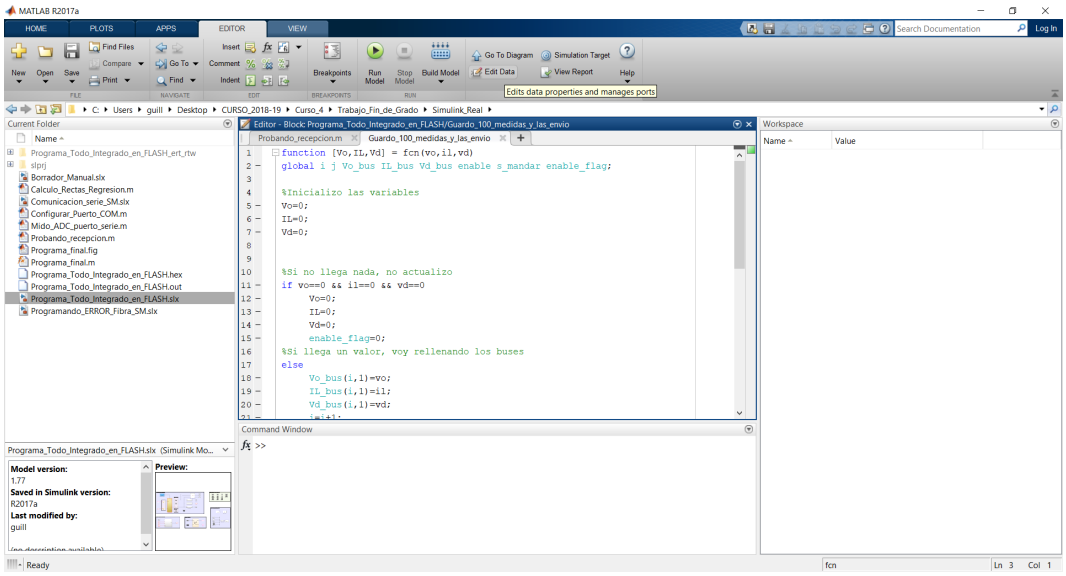


Figura 5-50. Diagrama Simulink Convertidor real. Área de memoria. Variable global en Bloque Matlab.

Se abrirá la ventana “Ports and Data Manager”, donde gracias a la herramienta “Add data”, se agregan tantas variables globales como sean necesarias en la función.

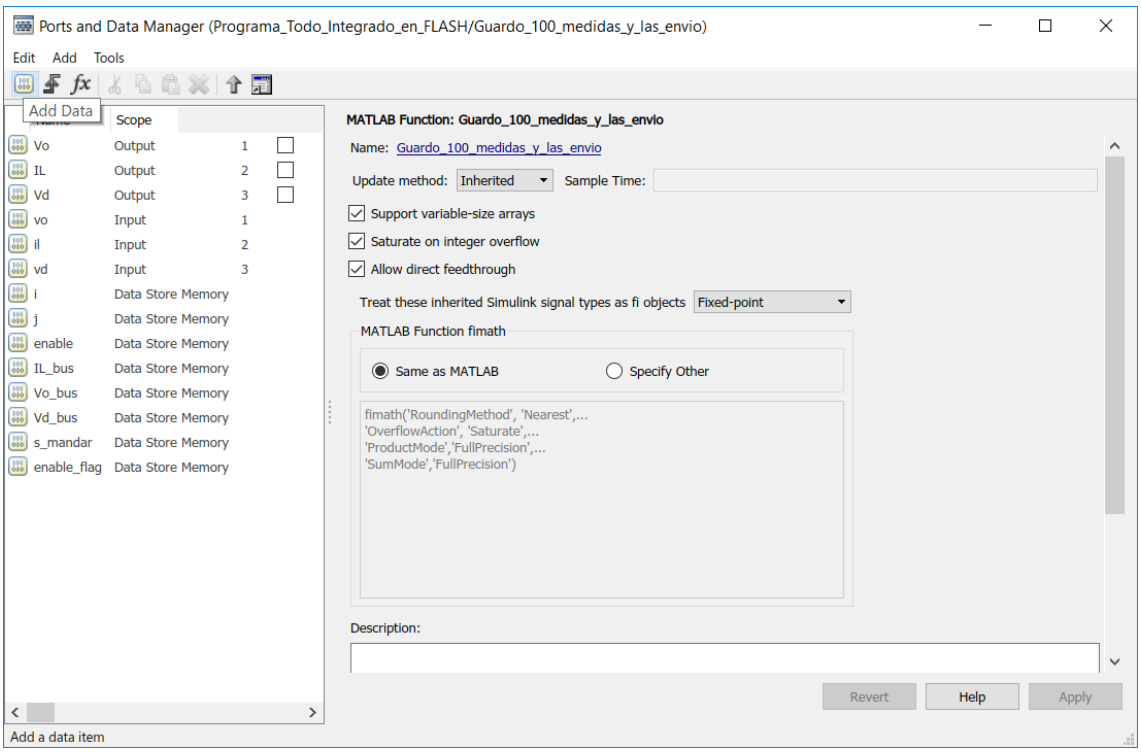


Figura 5-51. Diagrama Simulink Convertidor real. Área de memoria. Ports and Data Manager, Add data.

5.6. Interfaz GUIDE del proyecto.

5.6.1. Esquema Interfaz.

Una vez se encuentra el programa del convertidor instalado en el microcontrolador, se va a desarrollar una interfaz gracias a la cual sea posible poner en marcha, y comprobar el correcto funcionamiento del convertidor.

Para conseguir esto, se va a hacer uso de la herramienta de Matlab GUIDE, la cual con una gran versatilidad permitirá la comunicación y visualización del comportamiento del microcontrolador.

Para acceder a ella basta con introducir la orden 'guide' en Matlab.

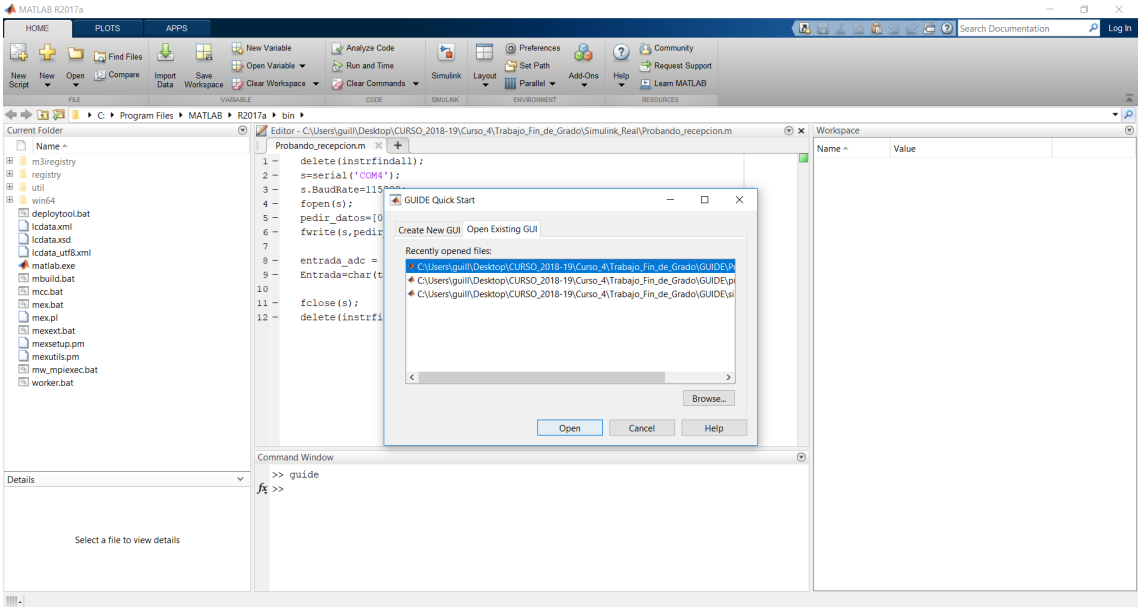


Figura 5-52. Diagrama Simulink Convertidor real. Herramienta guide.

Como ya se encuentra el proyecto creado, lo seleccionamos, si no basta con crear uno en blanco.

GUIDE permite diseñar una interfaz con numerosos elementos, con los que poder dar ordenes, representar con gráficas, escribir variables, dentro del programa. Por lo tanto, se analiza qué se quiere introducir en la ventana de desarrollo.

- Una gráfica donde representar los valores medidos de V_o, frente a V_d del ADC. Gracias a representar las dos, será posible observar su respuesta y evolución durante la ejecución del programa.
- Una gráfica donde representar los valores medidos de I_L del ADC.
- Un cuadro de mandos, con botones (actuadores), donde poner en marcha y parar el programa.
- Un cuadro de mandos, con los que enviar las órdenes de cambio de estados de funcionamiento del convertidor.
- Una o varias ventanas, que indiquen qué esta haciendo el convertidor, lo cual incluye desde conexión o desconexión, así como los estados del convertidor.

En el panel lateral izquierdo de GUIDE se encuentran todos estos elementos.

El proyecto de GUIDE ha quedado de la forma representada, donde se han incluido, además de las gráficas y actuadores descritos, unas ventanas de texto, con el objetivo de clarificar lo máximo posible el control.

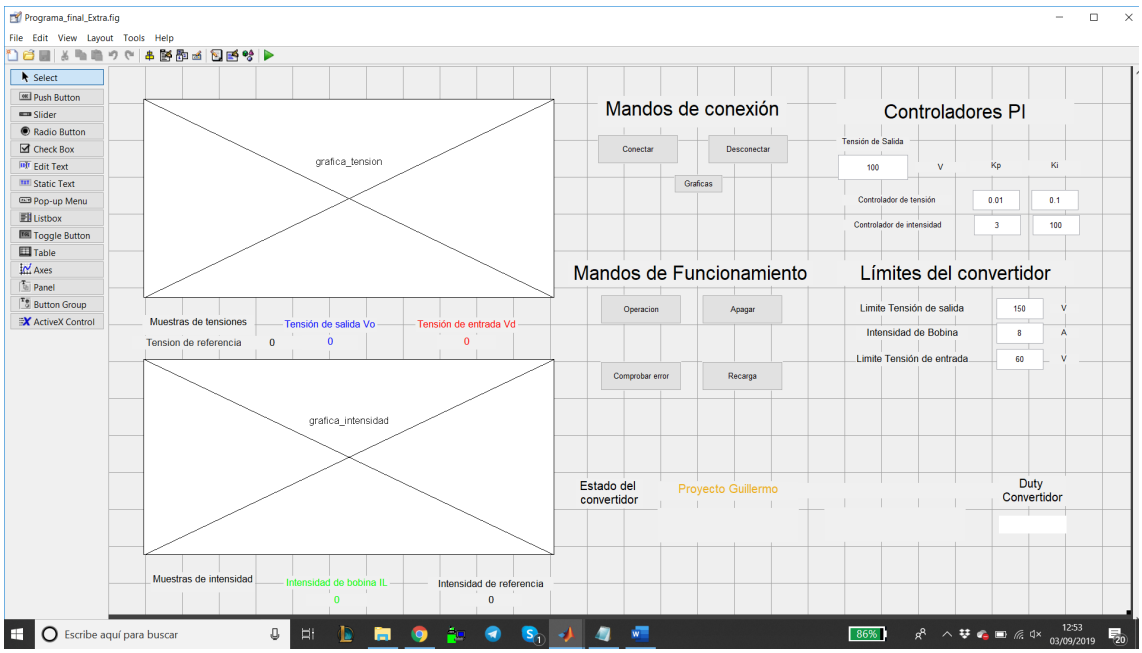


Figura 5-53. Diagrama Simulink Convertidor real. Herramienta guide. Proyecto Convertidor.

El diagrama se compone finalmente de:

- 2 Axes, donde representar las gráficas deseadas.
- 7 Push Botons, con los que mandar las órdenes de funcionamiento.
- 8 Edit Text, gracias a los cuales se podrán cambiar y enviar los parámetros del convertidor.
- 34 Static Texts, que pondrán nombre cada zona de la interfaz., así como mensajes sobre el estado del convertidor. Además, se usarán algunos para representar los valores numéricos de:
    - Tensión de referencia (V_{ref})
    - Tensión de salida (V_o) .
    - Tensión de entrada (V_d)
    - Intensidad de bobina (I_L).
    - Duty cycle del convertidor (D).

Los elementos cuando son introducidos, en el panel se nombran de forma estándar, como “axes_1”.

Si se quiere nombrarlos de otra manera, hay que hacer doble click con el botón izquierdo del ratón en cada elemento. Se abrirá la ventana Inspector, donde se podrá modificar diferentes campos como el nombre de objeto, el nombre con el que aparecerá en la simulación, color y tamaño de fuente.

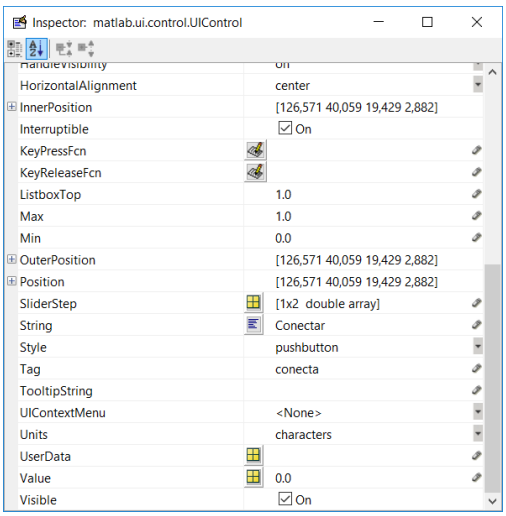


Figura 5-54. Diagrama Simulink Convertidor real. Herramienta guide. Ventana Inspector.

Recopilando la información de todos los objetos de GUIDE:

Elemento	Nombre Simulación (string)	Nombre de objeto (tag)
Gráfica de tensiones	–	*grafica_tension*
Gráfica de intensidad	–	*grafica_intensidad*
Botón conectar	*Conectar*	*conecta*
Botón desconectar	*Desconectar*	*desconecta*
Botón gráficas	*Graficas*	*grafica*
Botón operación	*Operacion*	*operacion*
Botón apagar	*Apagar*	*apagar*
Botón comprobar error	*Comprobar error*	*comprobar_error*
Botón recarga	*Recarga*	*recarga*
Texto Tensión de referencia	*Cambiará según funcionamiento*	*v_ref*
Texto Tensión de salida	*Cambiará según funcionamiento*	*Vo_cifra*
Texto Tensión de entrada	*Cambiará según funcionamiento*	*Vd_cifra*
Texto Intensidad de referencia	*Cambiará según funcionamiento*	*IL_error*
Texto Intensidad de bobina	*Cambiará según funcionamiento*	*IL_cifra*
Texto Duty Cicle	*Cambiará según funcionamiento*	*duty_cycle*

Texto de estado	*Cambiará según el estado*	*estado*
Texto de error	*Cambiará según el estado*	*tipoerr*
Texto de descripcion	*Cambiará según el estado*	*correcto*
Texto tipo error	*Cambiará según el estado*	*error*
Editor Tensión salida	*Cambia según el usuario*	*vo_ref*
Editor Kp tension	*Cambiará según el estado*	*PIvKp*
Editor Ki tension	*Cambiará según el estado*	*PIvKi*
Editor Kp intensidad	*Cambia según el usuario*	*PIiKp*
Editor Ki intensidad	*Cambiará según el estado*	*PIiKi*
Editor Vo máxima	*Cambiará según el estado*	*Limit_Vo*
Editor IL_máxima	*Cambiará según el estado*	*Limit_IL*
Editor Vd_máxima	*Cambia según el usuario*	*Limit_Vd*

Tabla 5-10. Nombre de los elementos de la interfaz de GUIDE (string y tag).

Una vez estén todos los elementos posicionados y correctamente nombrados, se procederá a guardar el proyecto. GUIDE crea dos archivos de Matlab:

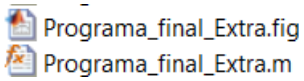


Figura 5-55. Diagrama Simulink Convertidor real. Herramienta guide. Archivos creados por Matlab.

- Programa_final_Extra.fig. Es la interfaz que se ha creado, en formato figura de Matalab. Previa programación no puede realizar ninguna acción, pero proporciona una visión de cómo queda la interfaz que se ha programado.
- Programa_final_Extra.m. Es el archivo de programación creado por Matlab. Aquí se va a programar qué acción se busca que realice cada elemento de la interfaz.

Figura 5-56. Diagrama Simulink Convertidor real. Herramienta guide. Ventana de Interfaz Convertidor.

5.6.2. Programación de la interfaz.

La programación de la interfaz creada por GUIDE se realiza en el archivo "Programa_final.m".

Este fichero, es el código de Matlab que ejecuta el cuadro de mandos que se ha diseñado, donde hay que precisar el tipo de funcionalidad y comportamiento de los elementos.

El código de este programa, "Programa_final.m" se ha aportado en los Anexos del proyecto. Sin embargo, se va a describir del mismo para ayudar a comprender su funcionalidad.

1. Programa_final_Extra_OpeningFcn.

Se trata de la función que se ejecutará al comienzo de ejecución del programa. En su ejecución imprime por pantalla unos mensajes preescritos.

2. Conecta_Callback.

Esta función se ejecutará cada vez que se presione el botón "Conecta".

Esta función inicializa las variables con las que trabajaremos más adelante, las cuales hay que declararlas como globales, puesto que van a ser utilizadas por distintas funciones del código.

Variable Global	Funcionalidad	Valor inicial
V_o	Vector que almacena 100 valores de V_o.	$zeros(1,100)$
I_L	Vector que almacena 100 valores de I_L	$zeros(1,100)$
V_d	Vector que almacena 100 valores de V_d.	$zeros(1,100)$
I_{ref}	Vector que almacena 100 valores de I_{ref}.	$zeros(1,100)$
$contador$	Vector que almacena 100 valores correspondientes al número de medidas.	$zeros(1,100)$
i	Variable índice para ejecución de programa.	1
j	Variable índice para ejecución de programa.	1
$estado$	Variable que indica el Estado en el que se encuentra el convertidor.	0
$pedir\_datos$	Variable bandera para habilitar la recepción de información por el puerto serie.	0

Tabla 5-11. Variables globales inicializadas en Programa_final_OpeningFcn.

Además de estas variables, se creaa un "timer", o reloj con las siguientes características.

Timer	Función de ejecución	Periodo de ejecución
t	$dibuja$	$0.001\ seg$

Tabla 5-12. Características del timer.

Con esta configuración, el reloj cuando se inicialice ejecutará la función $dibuja$, cada 1 milisegundo, que es equivalente a 1 kHz.

Se toman los valores que están escritos en los cuadros de adquisición de datos, y se envían al micro por el puerto serie. Para la transmisión de datos, se crea un vector de tamaño 10, "envio_micro", el cual será enviado por la conexión serie al microcontrolador.

Tras esto se actualiza el controlador con los nuevos valores reflejados en la interfaz.

Se resetean las gráficas para eliminar los datos previamente representados.

Una vez estén todas las variables inicializadas, se procede a realizar la conexión del programa con el puerto COMx. Esta conexión será de tipo serie a una velocidad de 115.200 $bauds$.

Hay que enviar una orden para inicializar el programa del microcontrolador y activar una bandera indicando que la conexión ha sido un éxito.

Se representa por el bloque de texto habilitado el estado.

3. Desconecta_Callback.

Esta función se ejecutará cuando sea pulsado el botón Desconecta.

En primer lugar, evalua si hay una conexión activa (variable bandera "encendido"). Si es así se procede a cerrar la conexión serie del puerto COMx.

Se elimina el timer "t", y se inicializan los índices.

Se representa por el bloque de texto habilitado el estado.

4. Gráfica_Callback.

Esta función se ejecutará cuando sea pulsado el botón Gráficas. En un principio se crea un transitorio de 3 segundos, donde son representados en el cuadro de texto tres puntos, a modo de espera.

Se actualiza la variable "pedir_datos", y es enviada gracias al vector "envio_micro".

En este momento se crea un vector mediante la función "fread", el cual guardará los primeros 100 valores que lleguen por el puerto serie.

Por último, se convierte este vector de tipo 'uint16' al tipo 'char', para evaluar esta información con los filtros diseñados.

Finalmente, se inicializa el timer "t", y se representa en el bloque de texto el estado 'Tomando medidas'.

5. Operación_Callback.

Esta función se ejecutará cuando sea pulsado el botón Operación.

En un primer lugar hay que evaluar si el convertidor elevador está encendido, con la variable bandera "encendido".

Se actualiza una variable global "estado", que indica que hemos dado la orden de pasar al estado "1", correspondiente a operación.

Siendo este el mismo comportamiento de las siguientes funciones, variando en la orden enviada por el puerto serie.

6. Apagar_Callback.

Esta función se ejecutará cuando sea pulsado el botón Apagar.

7. Recarga_Callback.

Esta función se ejecutará cuando sea pulsado el botón Recarga.

8. Comprobar_error_Callback.

Esta función se ejecutará cuando sea pulsado el botón Recarga.

Función ejecutada	Cambio de variables	Orden enviada
Graficas	$pedir\_datos = 1$	Al ejecutarse al comienzo, que corresponde al estado APAGADO, no hay transición de estados. Se comienza a representar los valores medidos en las gráficas
Operación	$estado = 1$	Pasar al estado de OPERANDO en la máquina de estado, pasando por PRECARGA y su transitorio. Seguir representando las gráficas.
Apagar	$estado = 3$	Pasar al estado de APAGADO en la máquina de estados, pasando por DESCARGAy su transitoiro. Seguir representando las gráficas.
Comprobar error	$estado = 7$	Pasar al estado de RECARGA en la máquina de estados, desde estado ERRORES. Seguir representando las gráficas.
Recarga	$estado = 8$	Pasar al estado de APAGADO en la máquina de estados, desde estado RECARGA. Seguir representando las gráficas.

Tabla 5-13. Órdenes enviadas por puerto Serie con botones del cuadro de mandos.

9. Dibuja.

Esta es la función que se ejecuta cada periodo de reloj, que corresponde a 1 milisegundo.

Hay que comprobar que la conexión del puerto serie está operativa con la variable bandera “encendido”.

En primer lugar, se evalua si la variable índice j es menos de 180. Si lo cumple significa que aún hay información en el vector “Entrada” que evaluar. Si no, se vuelve a tomar otra medida del puerto serie.

A la hora de evaluar la información, se pasan los filtros diseñados previamente:

- La cadena de lectura comienza, cuando se encuentren los caracteres de final ‘B’, y comienzo ‘A’ del mensaje de transmisión.
- Los datos recibidos se recomponen, para finalmente tener los valores en ‘uint16’, de la siguiente forma:

Comienza la cadena de descodificación de datos en el vector de tipo $char$. Se recorre este vector.

Una vez situado el inicio de la cadena, se asigna a cada variable, su valor recibido por el vector de enteros.

Aquí hay que prestar atención, ya que los valores enviados son de tipo 'uint16', el cual se fragmenta en dos en el envío de información.

Por lo tanto, el mapeo de las variables quedaría:

Variable recibida por comunicación serie.	Correspondencia con el vector de $'char'$
$'A'$	$Entrada(1)$
V_o	$[Entrada(2)Entrada(3)\ ]$
I_L	$[Entrada(4)Entrada(5)]$
V_d	$[\ Entrada(6)\ Entrada(7)]$
$Bandera\ 100\ medidas = ok\_proto$	$[\ Entrada(8)Entrada(9)]$
$Duty\_Cycle$	$[\ Entrada(10)\ Entrada(11)\ ]$
$Estado$	$[\ Entrada(12)\ Entrada(13)\ ]$
$Error$	$[\ Entrada(14)\ Entrada(15)\ ]$
I_{ref}	$[\ Entrada(16)\ Entrada(17)\ ]$
$'B'$	$Entrada(18)$

Tabla 5-14. Correspondencia de las variables recibidas por puerto serie con el vector char.

Gracias a la función de Matlab 'typecast' se realiza la conversión de estas variables en tipo 'uint16'

Entrada	$'uint8'$	$'uint16'$
$[char(1)\ char(2)]$	$[uint8(1)\ uint8(2)]$	$[uint\ 16]$

Tabla 5-15. Transformación por tipos de la información recibida por puerto serie.

- Si la variable ok_proto coincide con el valor 100, significa que en el microcontrolador hay 100 medidas guardadas de los ADC.
- Se procede a guardar los valores de los ADC en las variables globales dispuestas de medidas, y se siguen evaluando los vectores.
- Una vez se llegue a las 100 medidas de los tres módulos ADC, son representadas, V_o y V_d en la primera gráfica, e I_L en la segunda.

Nota: La variable I_{ref} se multiplica por 0.01, ya que viene 100 veces mayor que la real (se hizo la operación para facilitar la transferencia), y la variable del $Duty\_Cycle$ por 0.001.

El resto de funciones, se encargan de actualizar los valores de las variables que se pueden modificar.

Así cuando, por ejemplo, se aumenta el valor de la Tensión de referencia V_{ref}, o los componentes de los bloques PI del controlador, el programa envía dichos valores al micro.

5.6.3. Prueba de la interfaz.

Nota: Para el desarrollo de este punto, las capturas se tomaron con una versión primitiva de la interfaz durante su desarrollo, solo para comprobar que la base del programa funcionaba de forma correcta.

Con el fichero Programa_final.m completado, se puede ejecutar el cuadro de mandos programado.

Esta simulación se va a hacer teniendo en cuenta los siguientes puntos, y únicamente como ejemplo visual.

- En el diagrama de simulink, en el área de Interrupción, se añade el bloque de Negación (NOT) de la GPIO_In.

¿Por qué se ha hecho esto? El diagrama está preparado para tratar este error proveniente del Puente en H. Al no estar conectado, la salida está a '0' que tras la negación quedaría en '1'.

Al evaluar la máquina de estados, en el momento que se quiera pasar a un estado distinto de 0, se desvía al estado de ERROR, por lo que para este ejemplo se coloca el bloque NOT.

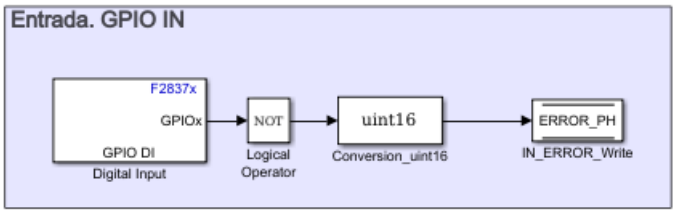


Figura 5-57. Diagrama Simulink Convertidor real. Bloque NOT de GPIO_in para probar interfaz.

Se conectar el pin correspondiente al $ADCV_o$ a 3,3 V, lo que equivale a 230 V medidos. Los otros dos pines de ADC, I_L y V_d, se van a a tocarlos con el dedo, lo que hará que midan un valor aleatorio.

1. Se pulsa el botón Conectar. En el cuadro "Estado del convertidor" ha cambiado el estado a Conectado. La conexión con el puerto serie COMx ha sido creada correctamente.

Figura 5-58. Diagrama Simulink Convertidor real. Puesta en marcha de la Interfaz. Pulsador Conectar.

2. Una vez creada la conexión del puerto serie, se inicializa la transferencia de información. Se pulsa el pulsador Gráficas, y las gráficas representan los valores medidos de los ADC (donde $V_d = 230\,V$, $V_o = 0\,V, I_L = 0A$).

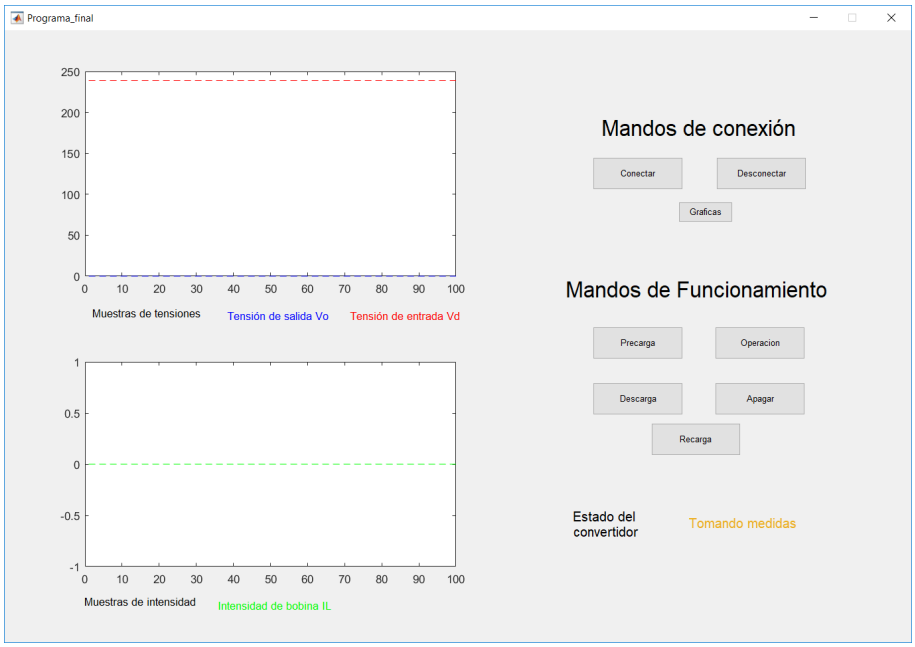


Figura 5-59. Diagrama Simulink Convertidor real. Puesta en marcha de la Interfaz. Pulsador Gráficas.

3. Para probar que los tres canales ADC son capaces de tomar medidas de forma simultánea, se implementa una tensión aleatoria en los canales de V_o e I_L.

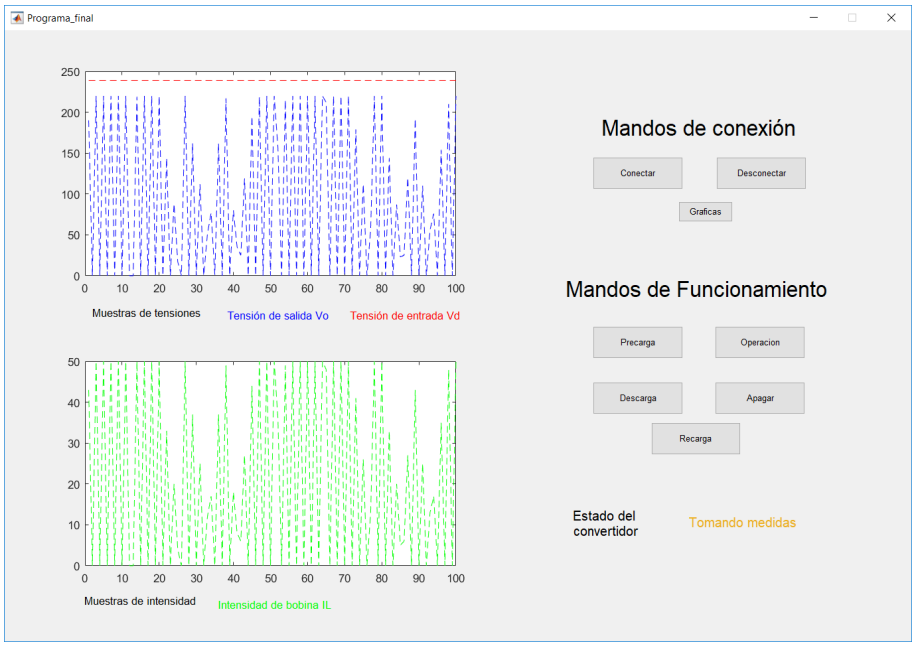


Figura 5-60. Diagrama Simulink Convertidor real. Puesta en marcha de la Interfaz. Canales ADC operando.

4. Una vez los ADC están tomando las medidas, se envian las órdenes de cambio de estado. Las gráficas seguirán actualizándose, y el cuadro de texto se actualiza al estado que pasa.

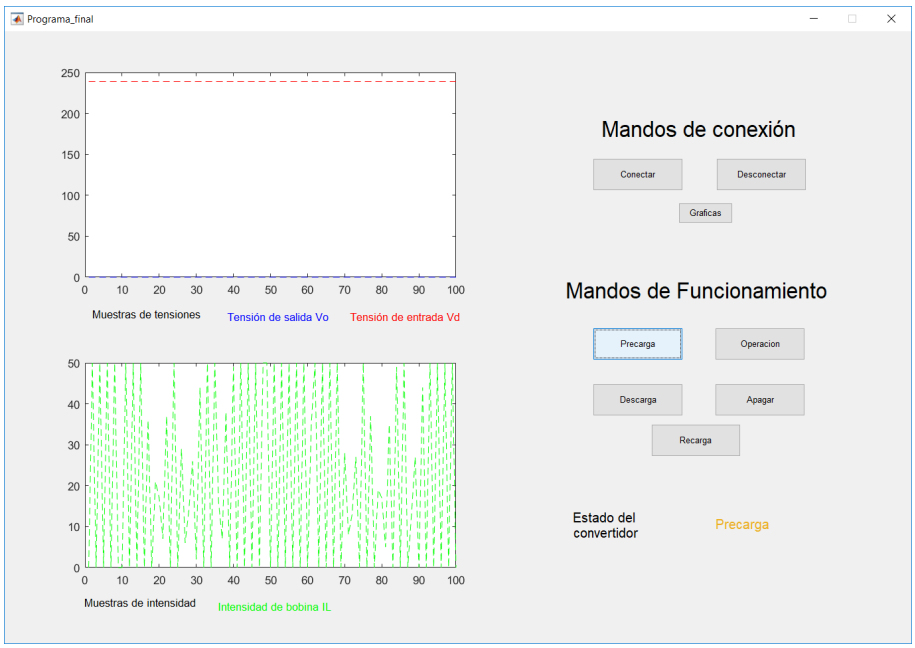


Figura 5-61. Diagrama Simulink Convertidor real. Puesta en marcha de la Interfaz. Envío de orden de cambio de Estado.

5. Para parar la simulación, se acciona el pulsador “Desconectar”, el cuál cerrará la conexión con el puerto serie COMx, parará la función timer y eliminará las variables con la que esté trabajando. Las gráficas se quedarán con la última tanda de medidas hasta que se reinicie la conexión.

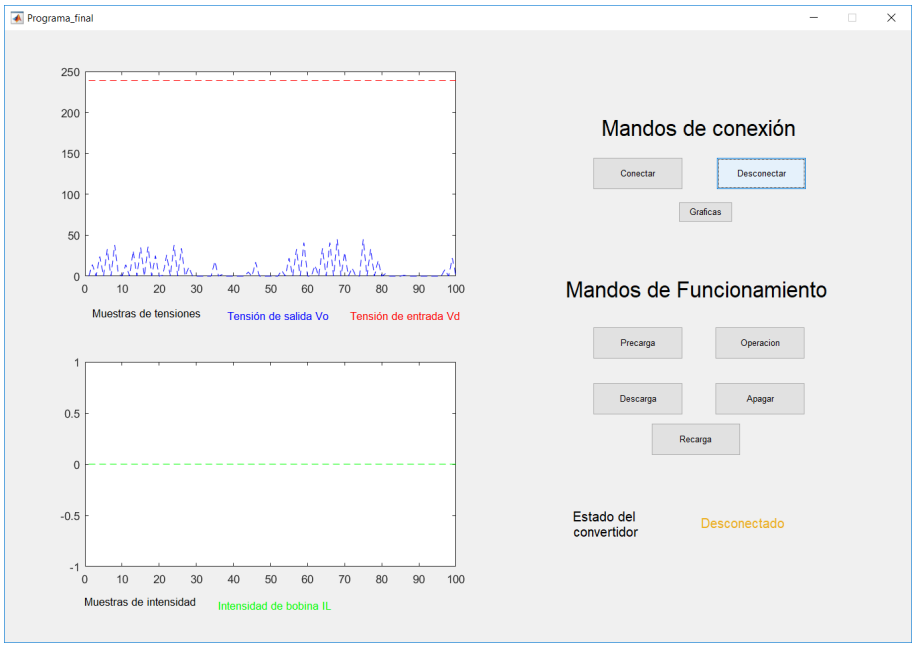


Figura 5-62. Diagrama Simulink Convertidor real. Puesta en marcha de la Interfaz. Desconexión de interfaz.

6 CONVERTIDOR ELEVADOR REAL

6.1. Circuito real elevador.

Una vez completada la programación del microcontrolador TMS320F28377S, así como la interfaz de control de Matlab, se procede a implementar el Convertirdor Elevador Real.

Para ello se va a emplear una serie de componentes electrónicos, los cuales ejercerán las funciones necesarias para alcanzar el objetivo de controlar el Convertidor.

Sea el diagrama del convertidor Elevador teórico.

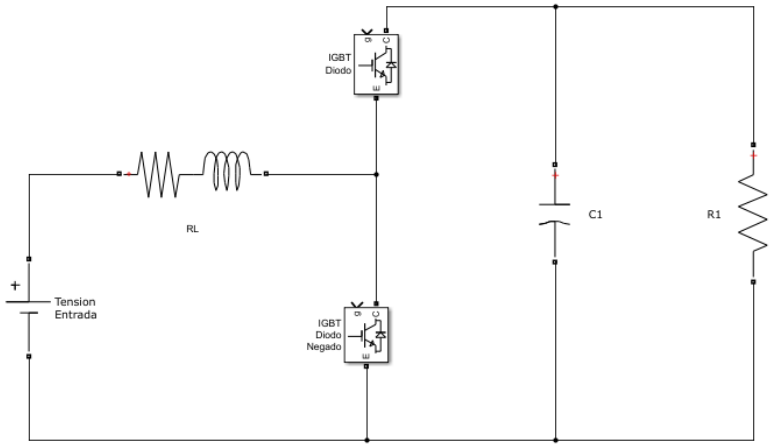


Figura 6-1. Diagrama convertidor Elevador Teórico.

El cuál, implementado por los elementos reales del circuito queda de la siguiente manera:

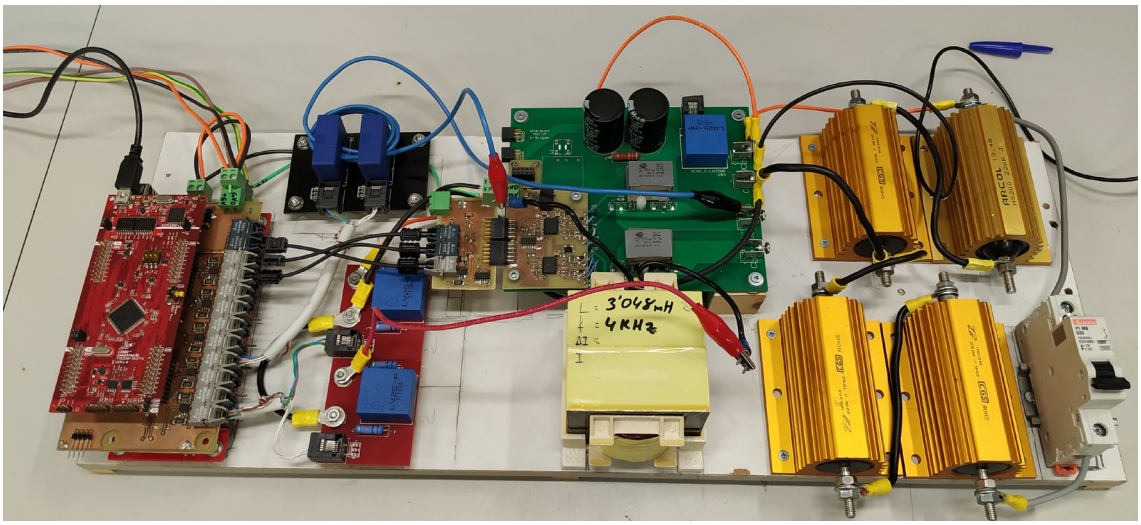


Figura 6-2. Convertidor Elevador construido.

6.2. Componentes Electrónicos.

Todos los circuitos y elementos electrónicos o eléctricos han sido proporcionados por el departamento de electrónica de potencia del departamento de la ETSI.

6.2.1. Circuito adaptador de la señal de PWM a fibra óptica.

Este circuito convierte la tensión a la salida del pin correspondiente al PWM que se ha programado, a una señal de fibra óptica.

Tiene la ventaja de necesitar una menor potencia, a la hora de transmitir el pulso, y mayor rapidez en el envio y recepción de información.

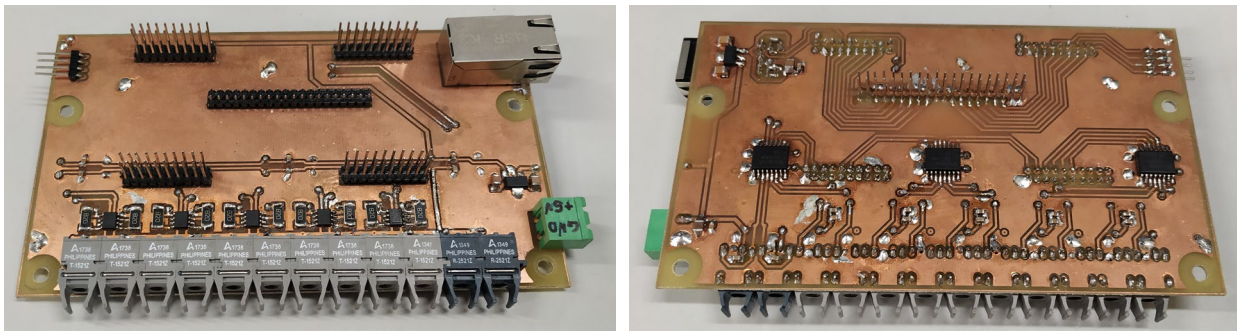


Figura 6-3. Circuito adaptador señal PWM a fibra óptica.

El circuito se conecta a la placa F28377S por la parte inferior de la misma, y se alimenta con una tensión continua de 5 V.

Dispone de terminales de recepción y otros de transmisión, lo cual habilita a enviar y recibir señales moduladas en el microcontrolador.

Se escogen dos de transmisión, para las señales PWM7_A y PWM7_B y uno de recepción correspondiente al ERROR del puente en H, como se ha visto en la programación.

<u>Canales de transmisión y recepción empleados.</u>

De este elemento, se va a necesitar los siguientes puertos de comunicación para implementar al circuito.

- Un canal de transmisión para envia la señal modulada del PWM. Puesto que la señal modulada del PWM7A corresponde al pin 40 del microcontrolador, y al primer terminal Tx del circuito.
- Un segundo canal de transmisión, para enviar señal de ENABLE para el componente Puente en H. Este trabaja con la señal modulada del PWM7B, pin 39 del microcontrolador, correspondiente al segundo terminal de transmisión Tx del circuito.
- Un canal de recepción, donde se recibirá una señal que indicará el estado de ERROR del Puente en H. Este corresponde a la GPIO 41 del DSP, que es el pin 5.

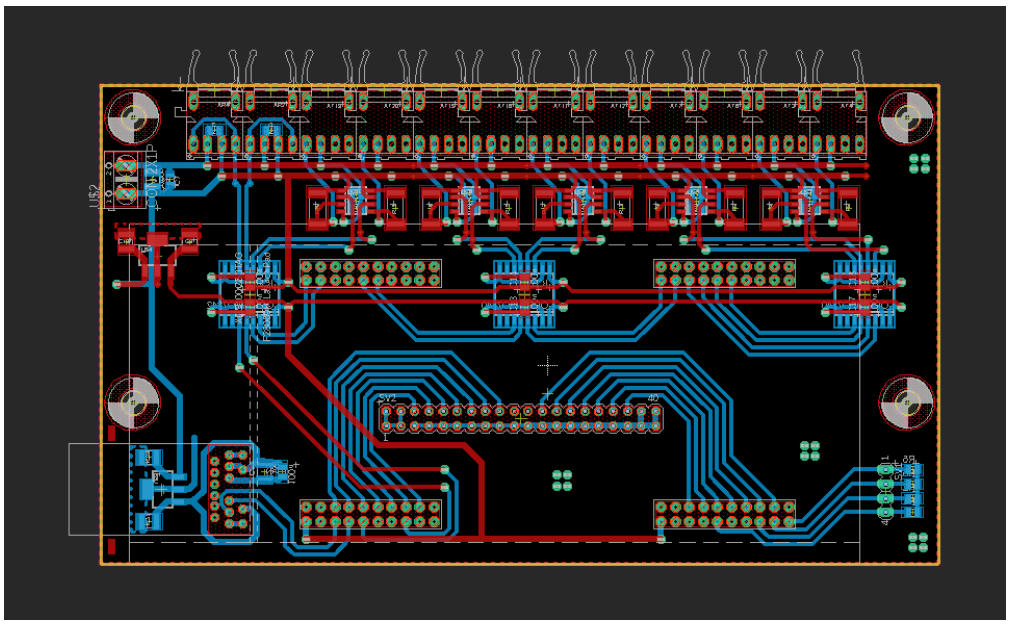


Figura 6-4. Circuito adaptador señal PWM a fibra óptica. Conexión de los pines del microcontrolador, a los terminales de recepción y transmisión de fibra óptica.

Inversión de la señal enviada por fibra óptica.

En los terminales de recepción Rx de fibra óptica se encuentra un Amplificador operacional, que invierte la señal recibe. Por esto, hay que tener en cuenta que la señal con la que se trabaje, va a convertirse en su opuesta cuando pase por los cables de fibra óptica.

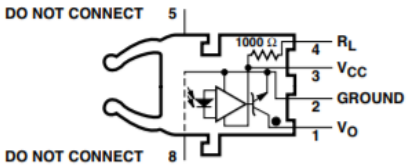


Figura 6-5. Circuito adaptador señal PWM a fibra óptica. Esquematico de terminal Rx del datasheet.

Teniendo esto en cuenta, se toman las siguientes medidas.

- La señal que transporta el Duty no se ve afectada.
- La señal de ENABLE (PWM7B) debe estar a nivel alto en el Puente en H. Puesto que su señal será invertida, hay que programarlo para que la tensión a la salida tenemos que programar la señal en el pin del microcontrolador a nivel bajo.
- La señal de ERROR se encontrará a nivel bajo cuando el Puente en H trabaje de forma correcta. Sin embargo, la GPIO del microcontrolador estará viendo un valor a nivel alto. Se tienen dos opciones para suplir este inconveniente:
    - Evaluar en la máquina de estados la transición de ERROR de forma contraria, hay error cuando la variable valga 0, y no lo hay cuando valga 1.
    - Añadir un bloque de negación, que invierta la señal de la GPIO de entrada. Esta es la que se encuentra implementada en el diagrama de simulink.

Señal en el microcontrolador	Sentido de señal en fibra óptica	Señal en Puente en H
1	→	0
0	→	1
0	←	1
1	←	0

Tabla 6-1. Valor de las señales transmirtidas entre el microcontrolador y el Puente en H, tras el paso por la fibra óptica

6.2.2. Circuito adaptador de los módulos ADC.

Tarjeta de adaptación, que adapta la señal de salida de los sensores LEM, de lectura de tensión e intensidad, de forma que pueda ser tratada por los terminales ADC del microcontrolador.

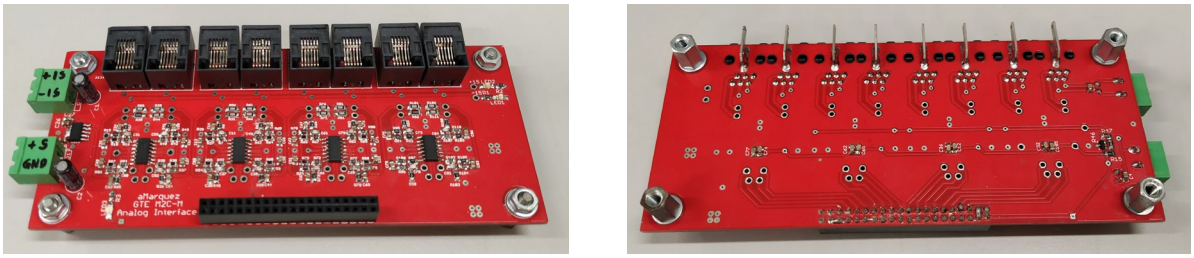


Figura 6-6. Circuito adaptador de señales medidas por sensores LEM a pines ADC.

El circuito se conecta a la placa F28377S a través del circuito de fibra óptica. Se encuentra situado en la base del conjunto de los tres circuitos.

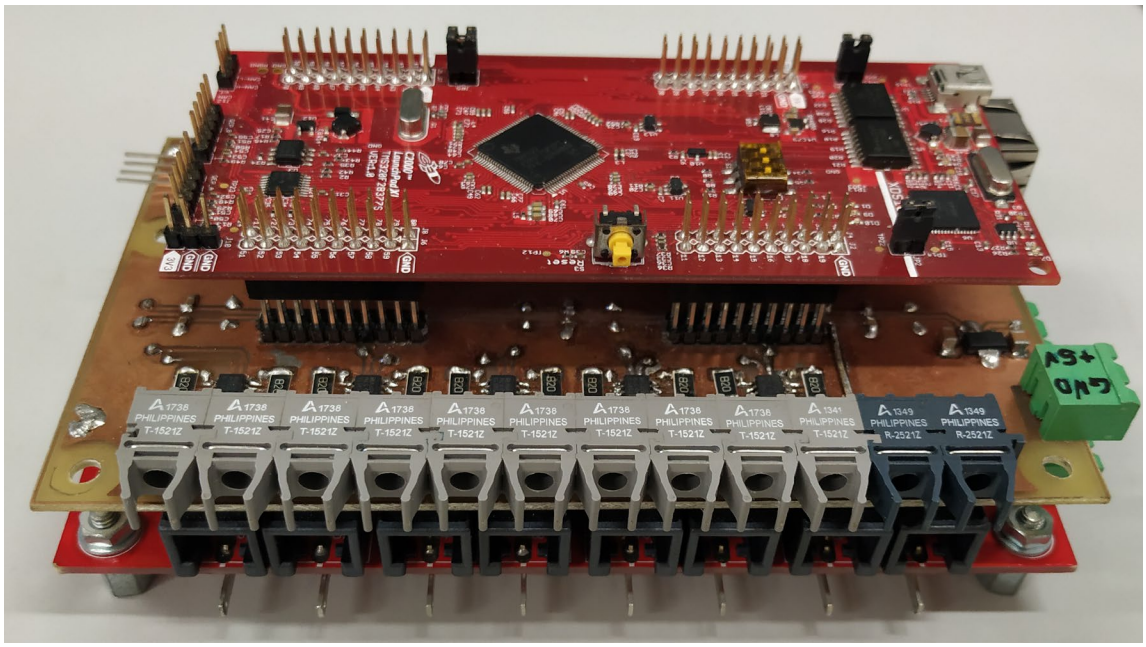


Figura 6-7. Circuito adaptador ADC. Conexión con microcontrolador y circuito de fibra óptica.

Hay que alimentar el circuito por dos terminales.

Terminal A	Terminal B
$\{GND, 5V\}$	$\{-15V, 15V\}$

Tabla 6-2. Terminales de alimentación del circuito adaptador ADC.

Etapa de adaptación.

El circuito, mediante la etapa de adaptación que se muestra, transforma la tensión dada por el sensor LEM de $\{0, 5V\}$, a una tensión a la que pueda trabajar el pin ADC del microcontrolador $\{0, 3.3\ V\}$.

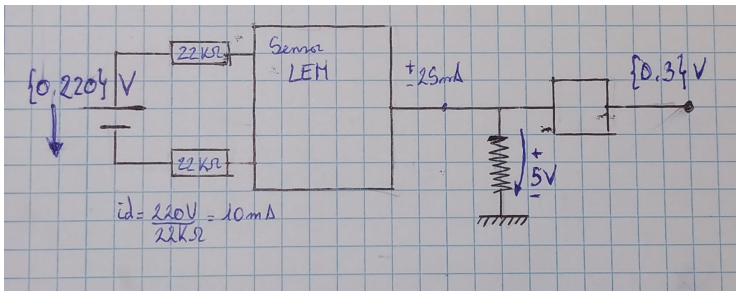


Figura 6-8. Circuito adaptador ADC. Etapa adaptadora desde sensor LEM a pin del microcontrolador.

Elección de los terminales del circuito.

El circuito dispone de 8 conectores, cada cual permite el acceso a dos sensores. Esto permite tomar medidas con hasta un total de 16 sensores LEM.

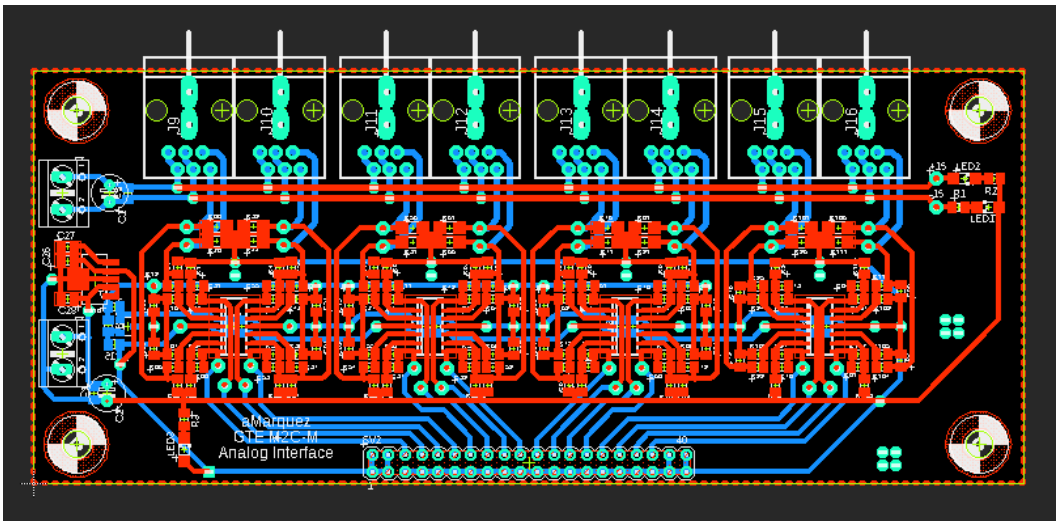


Figura 6-9. Circuito adaptador ADC. Esquemático de la tarjeta adaptadora.

La circuitería de los sensores, permite tomar distinto tipos de medidas.

- En algunos sensores, hay una realimentación negativa del amplificador con uno o más condensadores. Esto introduce una tensión de Offset en la medida, la cual, como se ha explicado en el apartado 5 permite tomar medidas que oscilen entre valores positivos y negativos.

Hay que adaptar unas señales de tensión, las cuales no tienen valor negativo, y una intensidad, que si necesita de un valor de offset para su cálculo. Por lo que se van a usar sensores sin offset para las tensiones, y sensor con offset para la intensidad.

(A) (B)

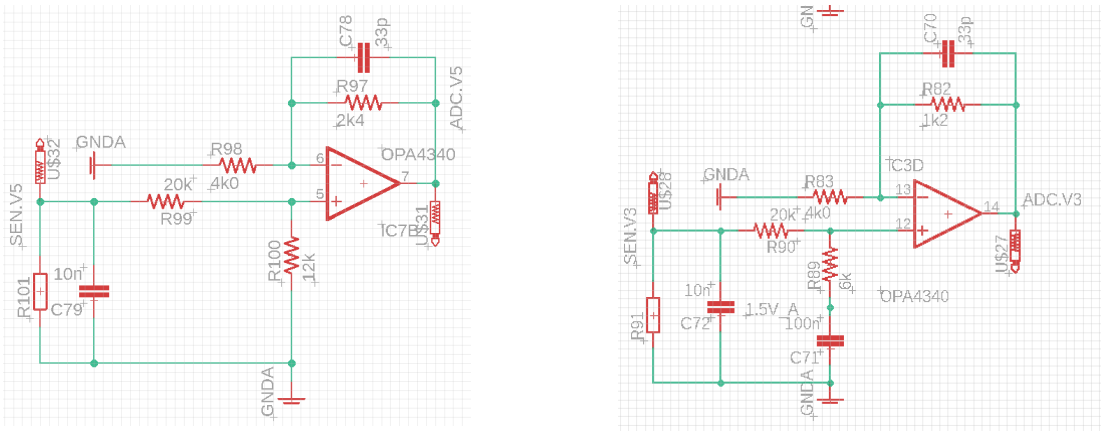


Figura 6-9. Circuito adaptador ADC. Esquemático de los sensores SEN8-7 (A) y SEN3 (B).

Por esto, se escogen los sensores del circuito:

Sensor tarjeta adaptadora.	Medida	Pin correspondiente F28337
SEN 3	I_L	68
SEN 7	V_d	63
SEN 8	V_o	64

Tabla 6-3. Correspondencia de los terminales del circuito adaptador ADC y pines del microcontrolador.

6.2.3. Sensores de tensión e intensidad.

En un primer momento, para tomar las medidas de tensión e intensidad del circuito, V_o, V_d e I_L, se van a emplear unos sensores LEM de tensión prefabricados para las pruebas. El sensor de intensidad será el mismo que empleemos.

Estos se cambiarán a la hora del montaje del circuito.

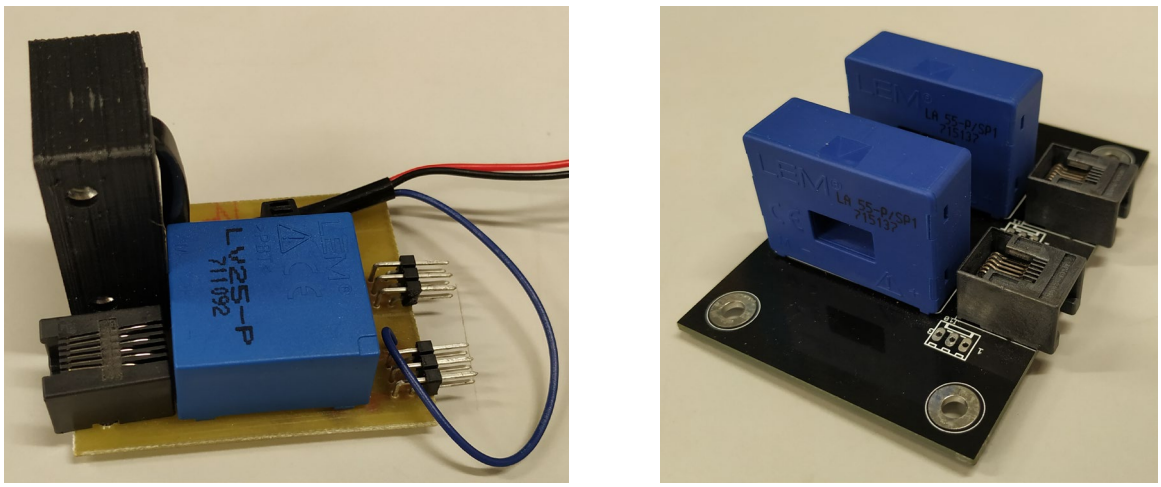


Figura 6-10. Sensores LEM de tensión (1) y corriente (2).

- En el sensor LEM de tensión, esta se mide entre los dos terminales con seis pines, los cuales están cortocircuitados. Necesitamos dos circuitos LEM de tensión para medir las dos tensiones.
- El sensor LEM de intensidad mide la corriente del cable que atraviesa el orificio del sensor. En la Figura 6-10. (2) hay dos sensores, se va a emplear uno de ellos.

  Para la hora de calibrar el sensor de intensidad, se va a enrollar el cable en el sensor, haciendo que lo atraviese más veces. Con esto se consigue que el sensor mida con una precisión mayor.

$$Corriente\ medida\ por\ sensor = Corriente\ cable * nº\ vueltas$$

Los sensores van conectados al circuito de adaptación ADC con los siguientes cables.

Figura 6-11. Cables sensores LEM.

Una vez esté montado el circuito elevador, se van a usar otros sensores de tensión que se han construido ensamblando los sensores LEM a dos resistencias cada uno.

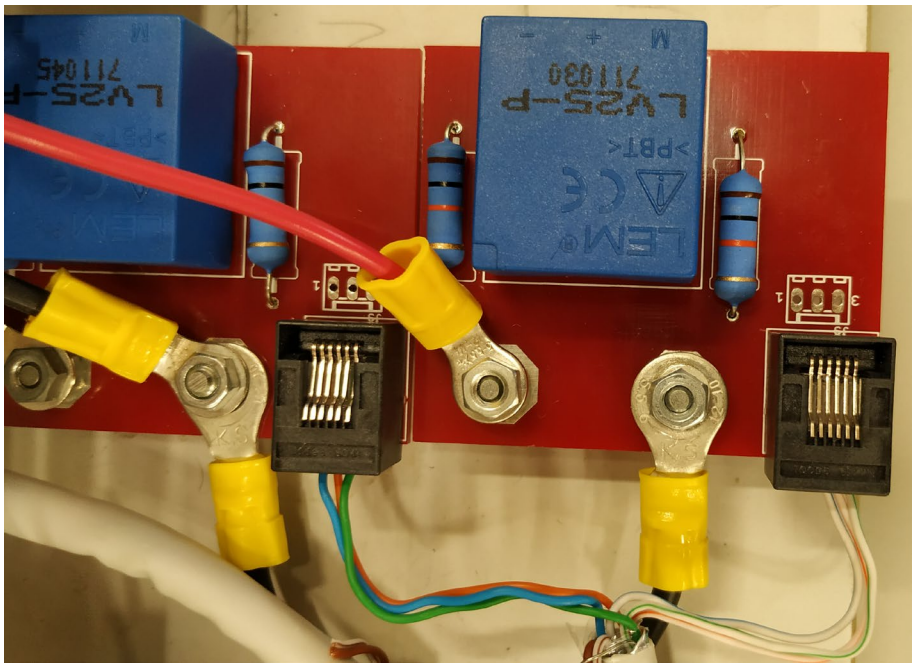


Figura 6-12. Sensores LEM de tensión del circuito.

Viendo como quedarían distribuidos los sensores LEM en el circuito final.

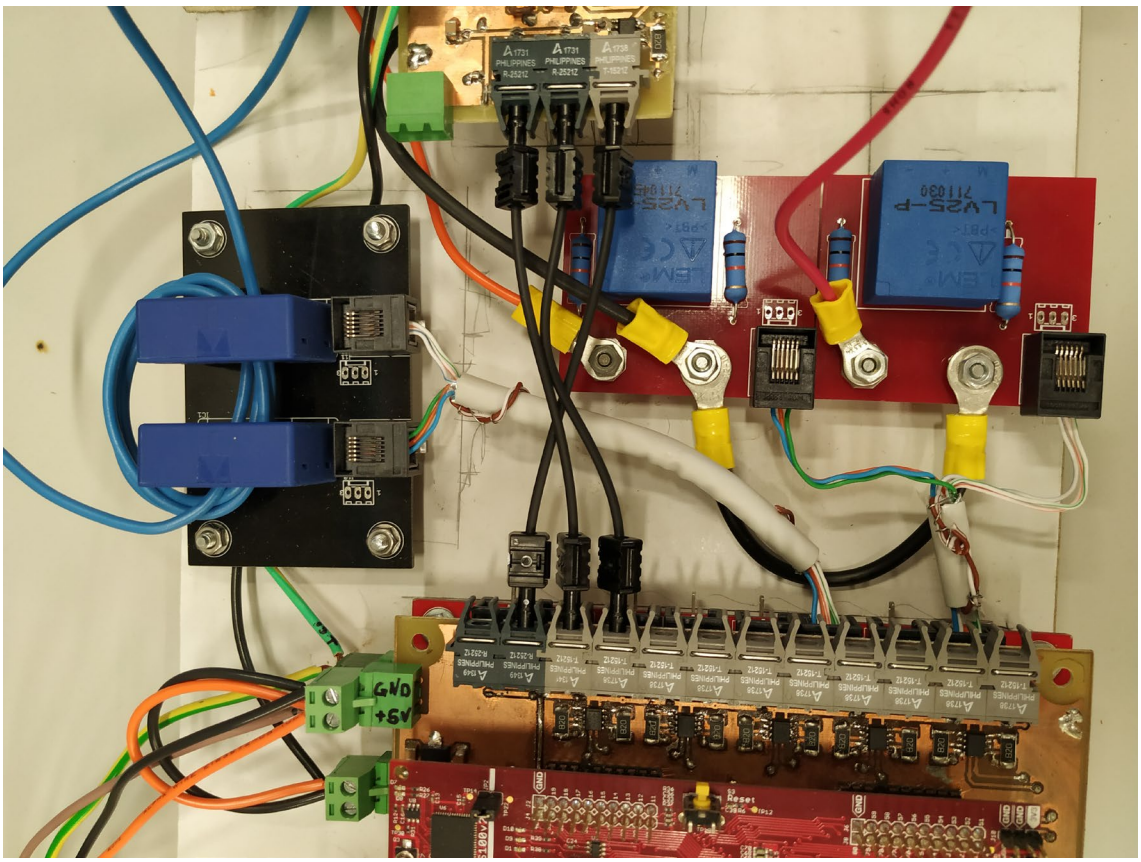
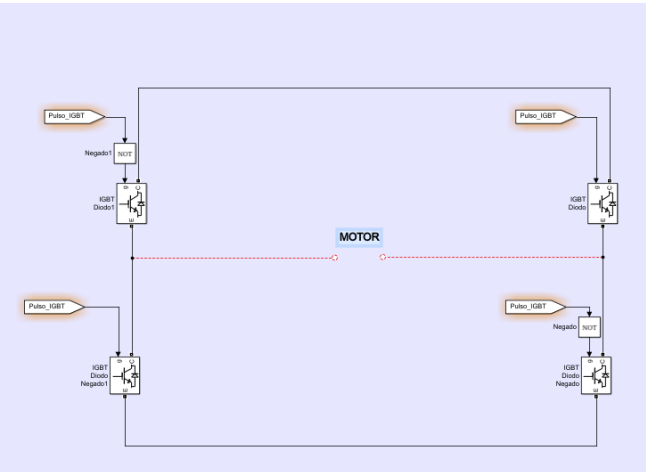


Figura 6-13. Vista de la posición de los sensores LEM de tensión del circuito.

6.2.4. Puente en H.

Un puente en H es un circuito formado por cuatro transistores, que operan emparejados por pares. Su uso más común es la puesta en marcha de motores de continua. En este proyecto, se va a utilizar una sola de las dos ramas de transistores, que equivalen a los dos interruptores de un circuito elevador.



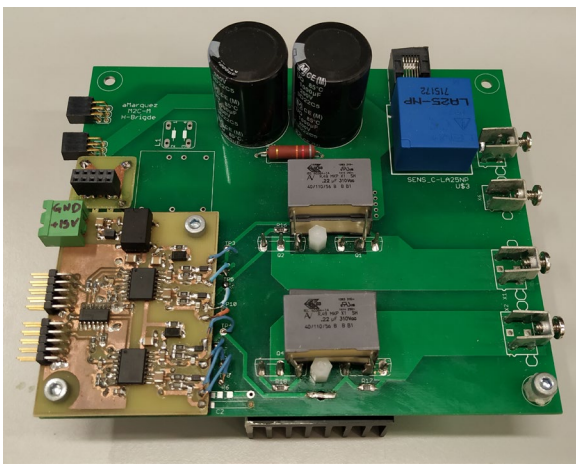


Figura 6-14. Diagrama puente en H (1). Circuito puente en H (2).

Representando el diagrama del puente en H ahora, de acuerdo a la construcción del dispositivo elevador se va a mostrar los nodos equivalentes al circuito, en el dispositivo electrónico.

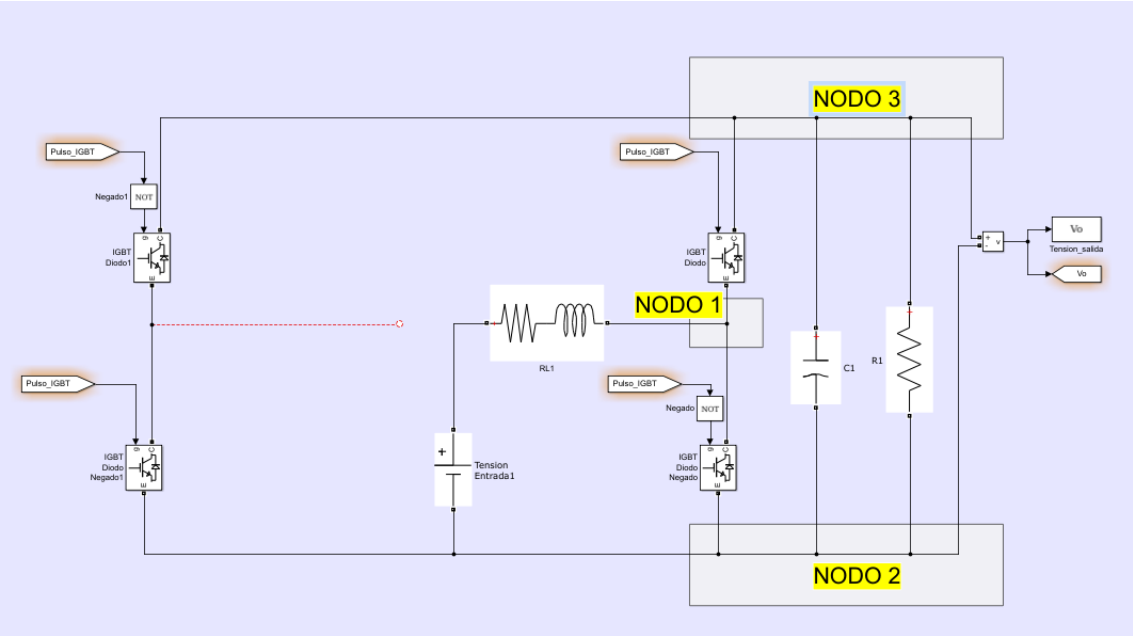


Figura 6-15. Diagrama del puente en H aprovechando una de las ramas para la implementación del convertidor Elevador.

Las cuales corresponderán con los pines del circuito real de la siguiente manera:

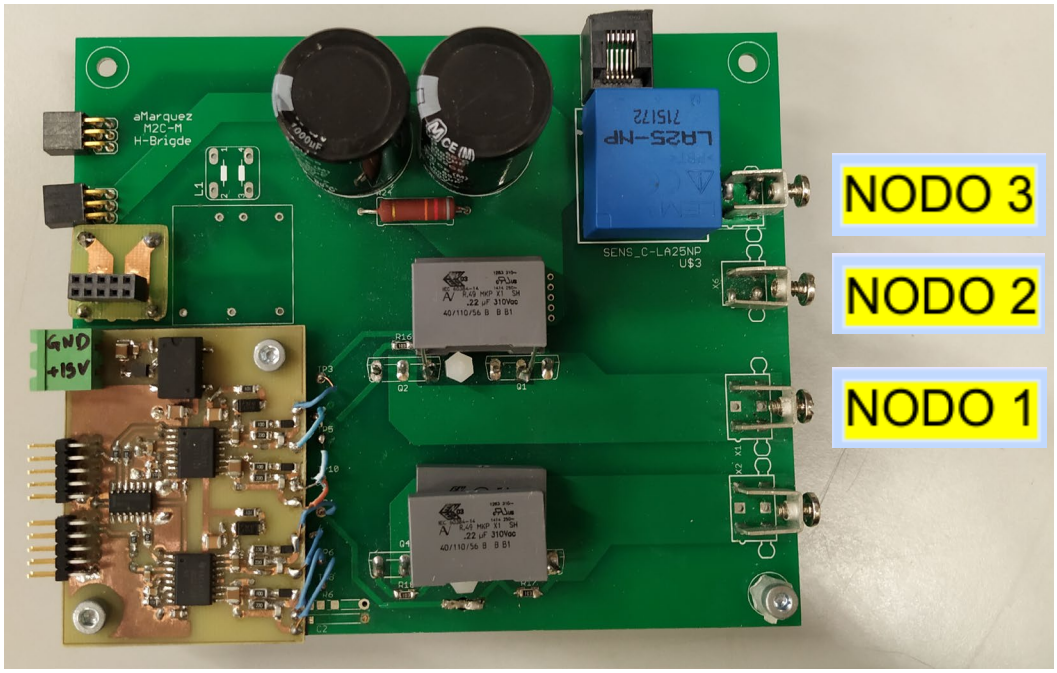


Figura 6-16. Correspondencias de los nodos del diagrama con los pines del Circuito.

El circuito de puente en H tiene una etapa de adaptación para fibra óptica, cuyos terminales corresponden:

Señal	Terminal en circuito adaptador a fibra óptica	Terminal en puente H
$PWM7\_A$	Transmisión (Tx)	Recepción (Rx)
$PWM7\_A$	Transmisión (Tx)	Recepción (Rx)
Error puente H	Recepción (Rx)	Transmisión (Tx)

Tabla 6-4. Correspondencia de terminales entre el circuito adaptador a fibra óptica y el puente en H.

Los límites eléctricos del circuito según los componentes son los siguientes:

Elemento	Límite
Interruptores $MOSFET$. Modelo: LRFP4227	$200\ V$; $65\ A$
Condensadores.	200 V

Tabla 6-5. Límites de los interruptores MOSFET.

6.2.5. Bobina.

La bobina del circuito, se encuentra localizada entre la fuente de tensión a la entrada del circuito, y la rama del puente en H. En este proyecto, ha sido suministrada por el departamento de Ingeniería Electrónica.

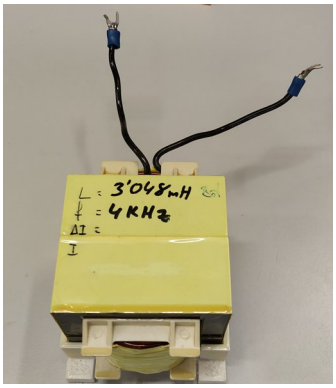


Figura 6-17. Bobina eléctrica del circuito.

Presenta las siguientes características:

Frecuencia de operación	$f = 4\ kHz$
Inductancia	$L = 3{,}048\ mH$
Límite de corriente	$I_{lim} = 20\ A$

Tabla 6-6. Correspondencia de terminales entre el circuito adaptador a fibra óptica y el puente en H.

6.2.6. Batería de resistencias.

El dispositivo puede operar con la salida en vacío, esto es, sin ninguna resistencia a la salida. Sin embargo, en sus aplicaciones reales se va a producir una potencia a la entrada y a la salida del convertidor.

$$P_{entrada} = P_{salida}$$

Esto se consigue añadiendo una serie de resistencias conectadas a la tensión de salida.

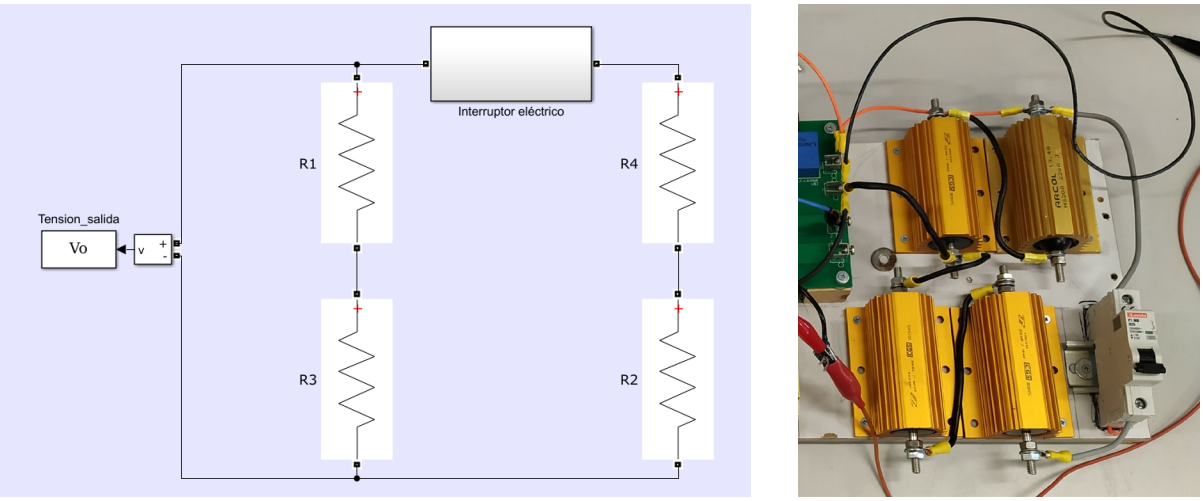


Figura 6-18. Diagrama mostrando la batería de resistencias conectadas a la salida del Elevador.

Sean los valores de las resistencias:

$$R_1 = R_2 = R_3 = R_4 = 220\ \Omega$$

Como en cada rama hay dos resistencias en serie, el valor de la resistencia por rama será:

$$R_{rama} = 2 * R = 440\ \Omega$$

Si se conectasen las dos ramas activando el interruptor, están en paralelo las ramas, obteniendo:

$$R_{total} = \frac{R_{rama} * R_{rama}}{R_{rama} + R_{rama}} = 220\ \Omega$$

Haciendo los cálculos, tomando datos reales de los experimentos.

$$P_{entrada} = V_d * I_L = P_{salida} = V_o * I_o = \frac{V_o^2}{R_{salida}}$$

Si a la salida hay una tensión de $100\ V$, a la entrada $50\ V$, los valores teóricos serán:

Número de resistencias	P	I_L	I_o
Una rama de resistencias	$22{,}73\ W$	$0{,}45\ A$	$0{,}227\ A$
Ambas ramas	$45{,}45\ W$	$0{,}91\ A$	$0{,}45\ A$

Tabla 6-7. Valores teóricos de la Potencia en las resistencias e intensidades en la bobina y a la salida del convertidor.

6.2.7. Interruptor magnetotérmico.

Para conectar las dos ramas de resistencias del apartado anterior se emplea un interruptor magnetotérmico, de la marca Lobato, con las siguientes características eléctricas.

Serie	$P1MB$
Número de polos	1
Corriente nominal	$6\ A$
Características de disparo	$Tipo\ C$
Tensión AC Nominal	$230\ V$
Tensión DC Nominal	$60\ V$
Capacidad de Ruptura	$10\ kA$

Tabla 6-8. Características eléctricas del interruptor magnetotérmico.

6.2.8. Fuentes de alimentación.

Alimentación de los componentes del circuito.

Las distintas placas que componen el circuito real requieren una alimentación determinada.

Circuito	Alimentación
Etapa adaptadora de módulos PWM	$[GND, +5\ V]$
Etapa adaptadora ADC	$[GND, +5\ V]\ ;\ [-15\ V, +15\ V]$
Puente en H	$[GND, +15\ V]$

Tabla 6-9. Alimentación de los componentes electrónicos del elevador.

Se va a emplear la siguiente fuente de tensión, suministrada por el departamento de ingeniería electrónica. Esta fuente aporta unas salidas a $GND, +5\ V, -15\ V$ y $+15\ V$.

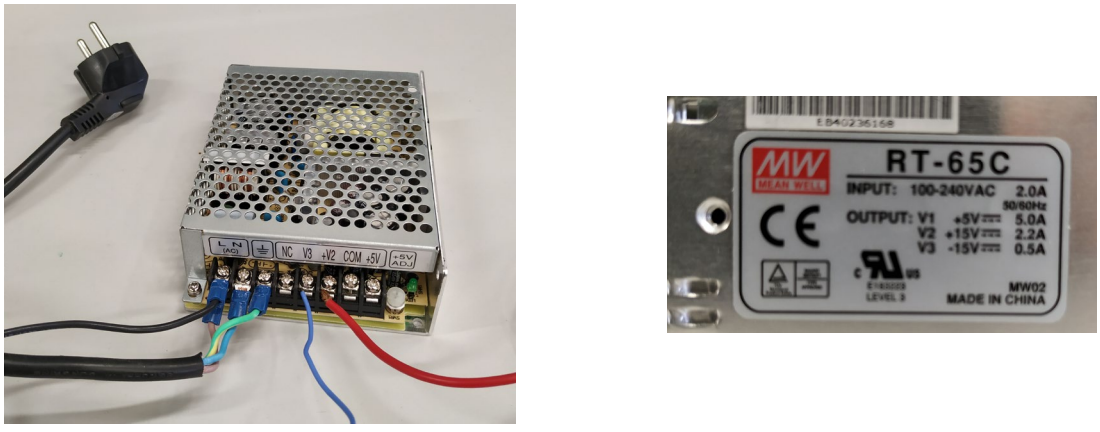


Figura 6-19. Fuente alimentación componentes.

Alimentación del convertidor elevador.

Se va a hacer uso de otra fuente de tensión, con la que se aportará la tensión de entrada al circuito V_d. Las características de la fuente son para nuestro circuito:

Tensión.	$[0, 220\ V]$
Corriente	$[0\ A, 2.725\ A]$

Tabla 6-10. Salidas de la fuente de tensión que alimenta el convertidor.

Nota: 220 V es lo máximo que pueden medir los sensores de tensión.

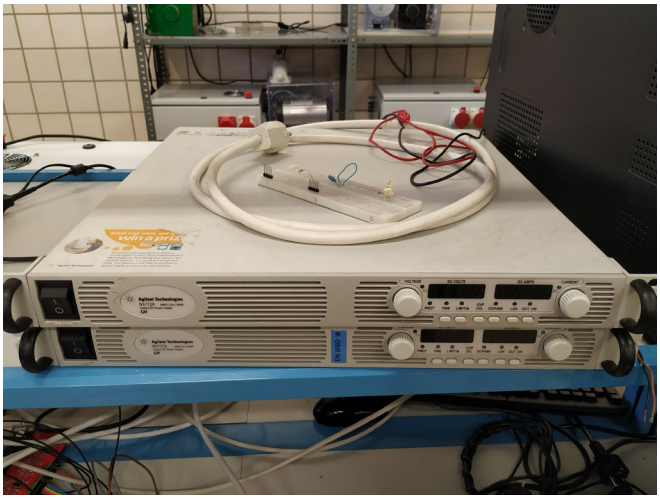


Figura 6-20. Fuente alimentación circuito elevador.

Se puede colocar el dispositivo elevador encima de la fuente de tensión, con lo que ahorrando así espacio.

Figura 6-21. Fuente alimentación V_d junto con el circuito elevador.

La conexión de los terminales será:

- Cable rojo a la bobina, ya que es el terminal positivo de la fuente de tensión.
- Cable negro a la tierra del circuito.

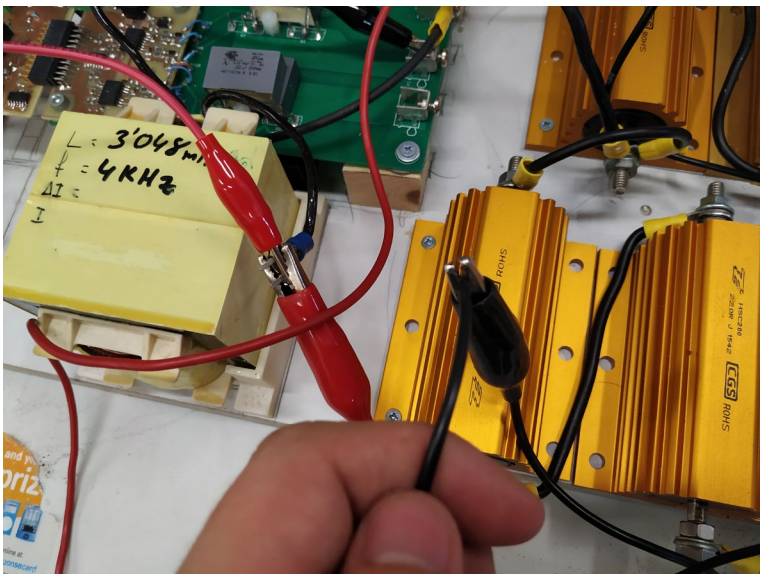


Figura 6-17. Conexiones de los terminales de la fuente de alimentación V_d.

7 PUESTA EN FUNCIONAMIENTO DEL CIRCUITO

Una vez está todo el circuito, así como la programación completa, se va a realizar una serie de experimentos con el elevador con los que se analizarán distintas características del mismo.

7.1. Puesta en marcha de la interfaz.

Para comenzar a realizar los experimentos se pone en marcha la interfaz. Con la ayuda de un osciloscopio, con dos sondas, se va a medir, con una sonda de tensión con la que se medirá la tensión de salida V_o y otra de intensidad, con la que será medida la intensidad de la bobina I_L.

Se comienza ejecutando el archivo 'Programa_final_Extra', que incluye todas las medidas, comandos y opciones que se han implementado en la interfaz.

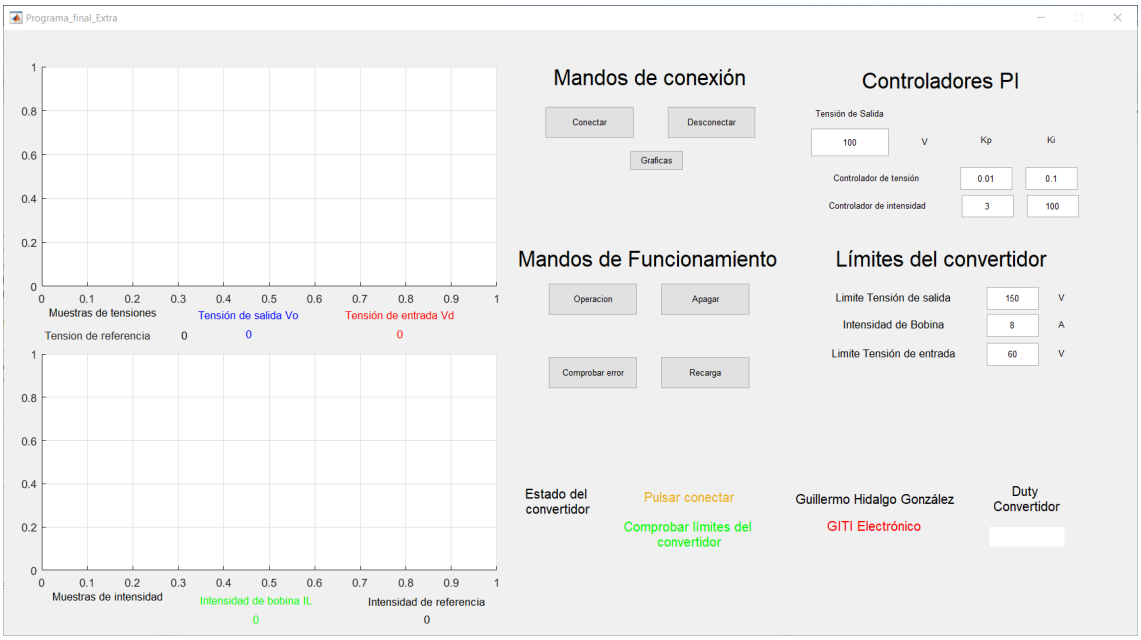


Figura 7-1. Ventana de comandos de la Interfaz del proyecto.

Comprobando:

- Los parámetros expresados en la interfaz de control son los correctos, esto incluye:
  - Límites de tensiones e intensidad.
  - Componentes de los controladores PI del programa.
  - La tensión de salida deseada será la que queramos en cada experimento.
- Por la parte física, hay que comprobar la buena conexión de todo el circuito, incluyendo las placas adaptadoras bien alimentadas, y que los terminales están bien conectados al sitio que deben.

Se pulsan los botones de conectar y representar las gráficas.

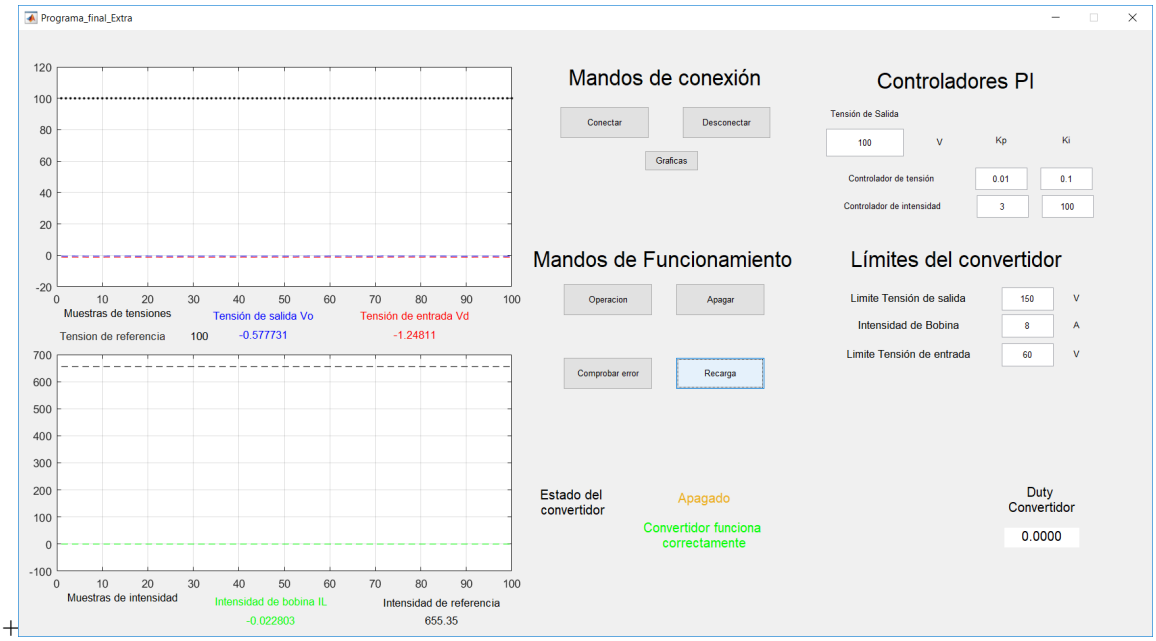


Figura 7-2. Ventana de comandos. Inicialización de la interfaz.

Conectada la fuente de tensión de entrada, se observa que comienza a medir de forma correcta las tensiones y la intensidad, tanto en las gráficas, como en sus valores numéricos.

Figura 7-3. Ventana de comandos. Iniciando toma de medidas.

7.2. Primera prueba.

En esta prueba se impone una tensión de salida de $V_o = 100\ V$. Además, se alimenta el circuito con una tensión de entrada de $V_d = 50\ V$, usando una de las ramas de resistencias.

V_d	50 V	Duty teórico	0,5
V_o	100 V	Resistencias	1 ramas (440 Ω)

Tabla 7-1. Parámetros experimento 1.

Se pulsa el botón de operación y se analizan las respuestas del convertidor.

- En un primer lugar, la intensidad de referencia cae bruscamente. Esto es debido a que el control comienza a operar, y el primer PI (correspondiente al de tensión) calcula dicha referencia.
- La tensión sufre una sobreoscilación, debido al cambio brusco originado en el control. Habría que probar distintas configuraciones de los parámetros de los bloques PI, para rebajar dicha sobreoscilación.
- El duty cycle tiende a un valor próximo al teórico.

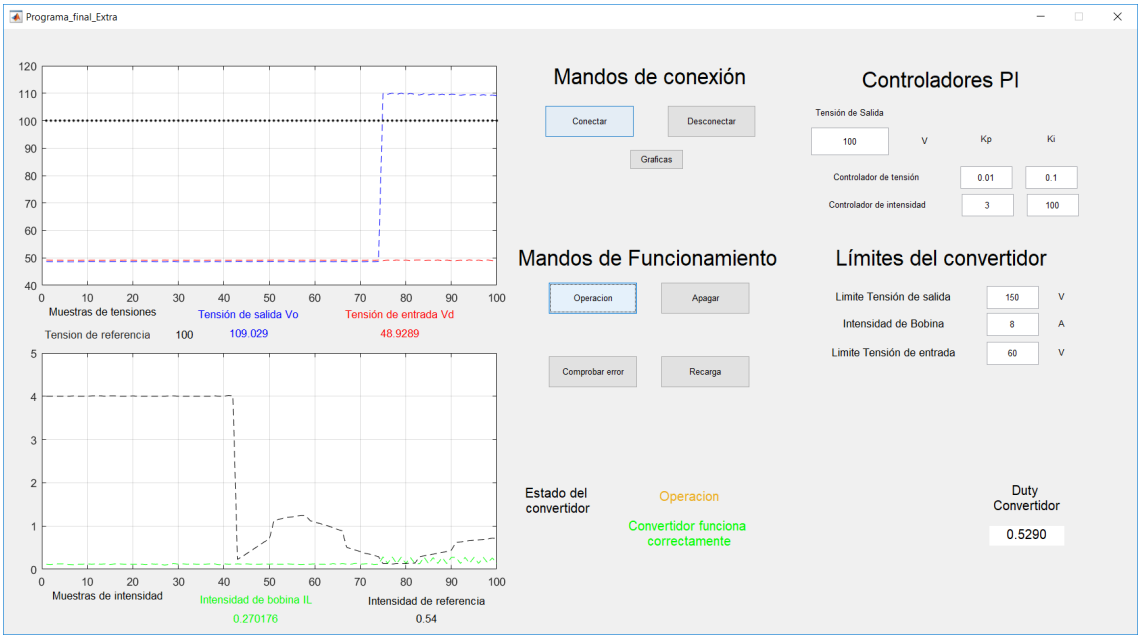


Figura 7-4. Primera prueba. Puesta en marcha de la ventana de comandos.

Tras un breve transitorio, las medidas se estabilizan.

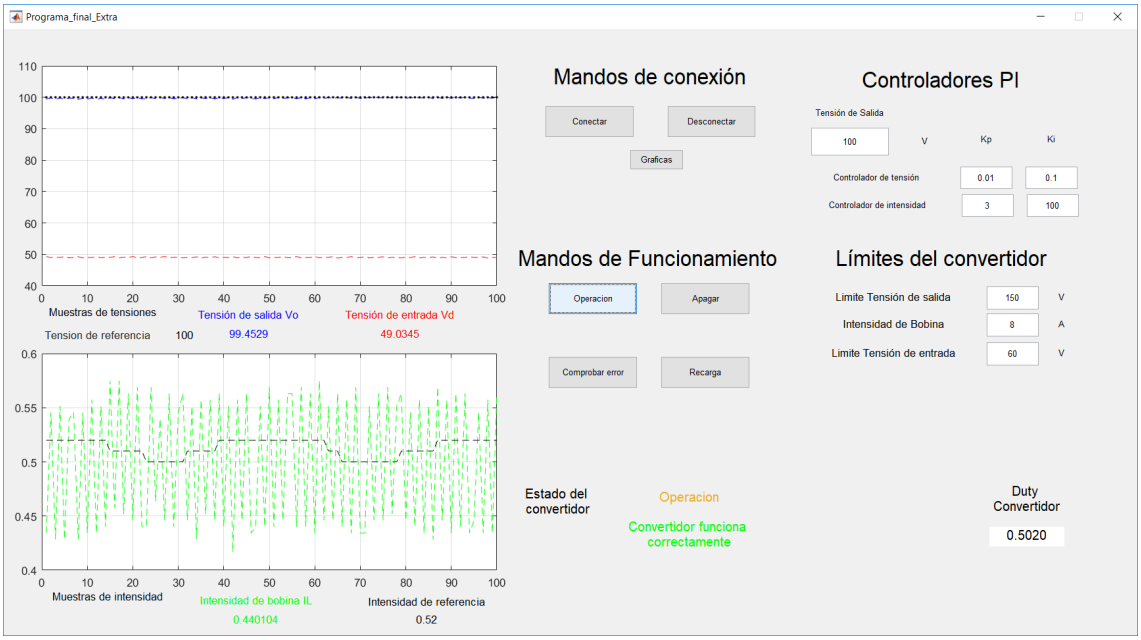


Figura 7-5. Primera prueba. Medidas de la ventana de comandos.

Se ha estudiado al mismo tiempo, la respuesta del circuito real gracias a un osciloscopio. Se ha medido la intensidad en la bobina I_L gracias a una sonda de intensidad, y la tensión a la salida V_o con una sonda de tensión.

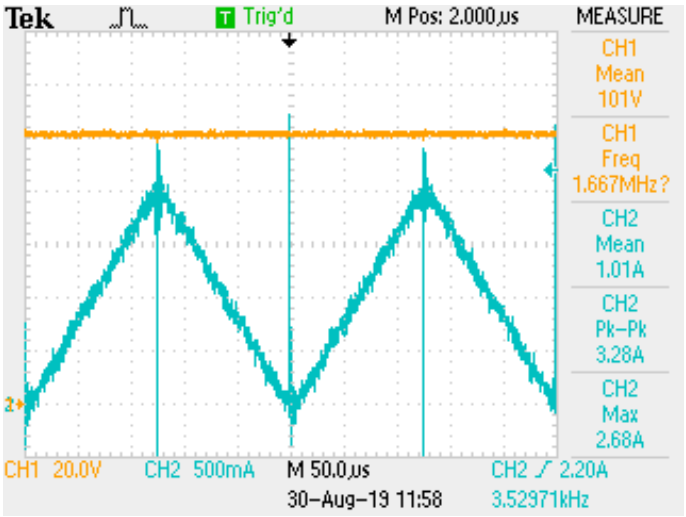


Figura 7-6. Primera prueba. Medidas tomasdas por osciloscopio.

7.3. Segunda prueba.

En esta prueba se impone una tensión de salida de $V_o = 60\ V$. Además, se alimenta el circuito con una tensión de entrada de $V_d = 50\ V$, usando una de las ramas de resistencias.

V_d	50 V	Duty teórico	0,16667
V_o	60 V	Resistencias	1 rama (440 Ω)

Tabla 7-2. Parámetros experimento 2.

Los resultados de la segunda prueba son:

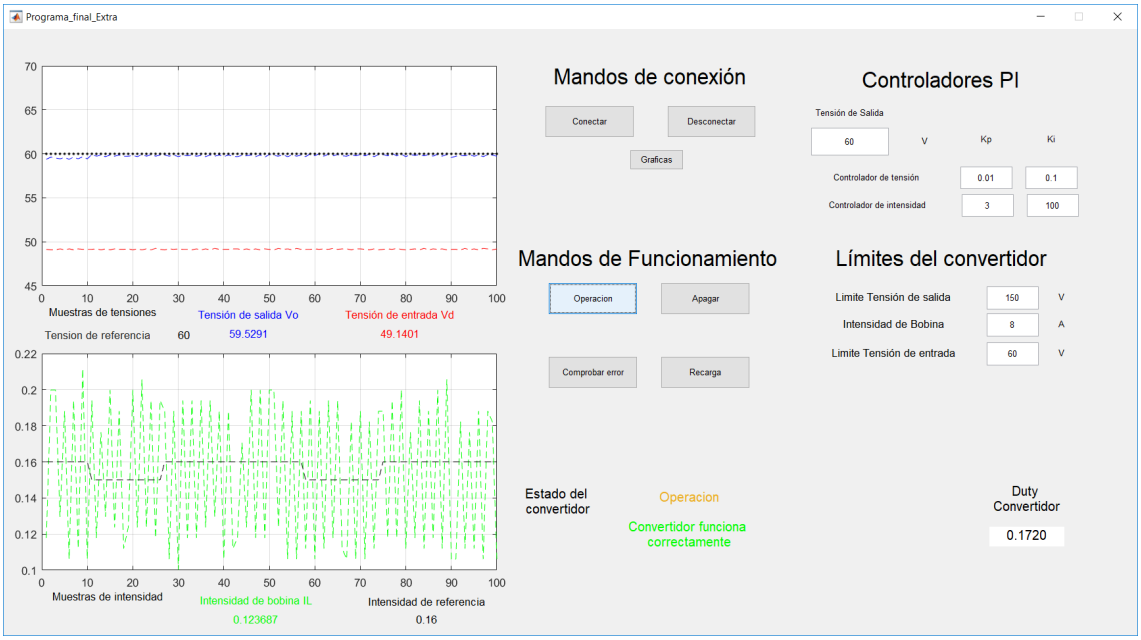


Figura 7-7. Segunda prueba. Medidas de la ventana de comandos.

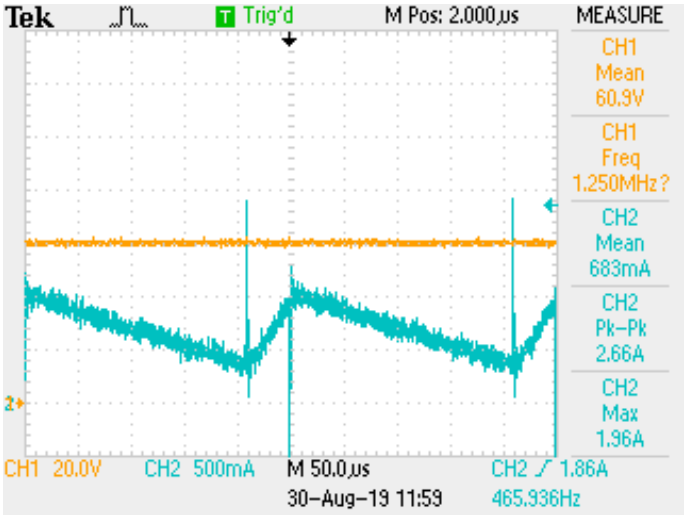


Figura 7-8. Segunda prueba. Medidas tomasdas por osciloscopio.

7.4. Tercera prueba.

En esta prueba se impone una tensión de salida de $V_o = 160\ V$. Además, se alimenta el circuito con una tensión de entrada de $V_d = 50\ V$, usando una de las ramas de resistencias

V_d	50 V	Duty teórico	0,6875
V_o	160 V	Resistencias	1 rama (440 Ω)

Tabla 7-3. Parámetros experimento 3.

Los resultados de la tercera prueba son:

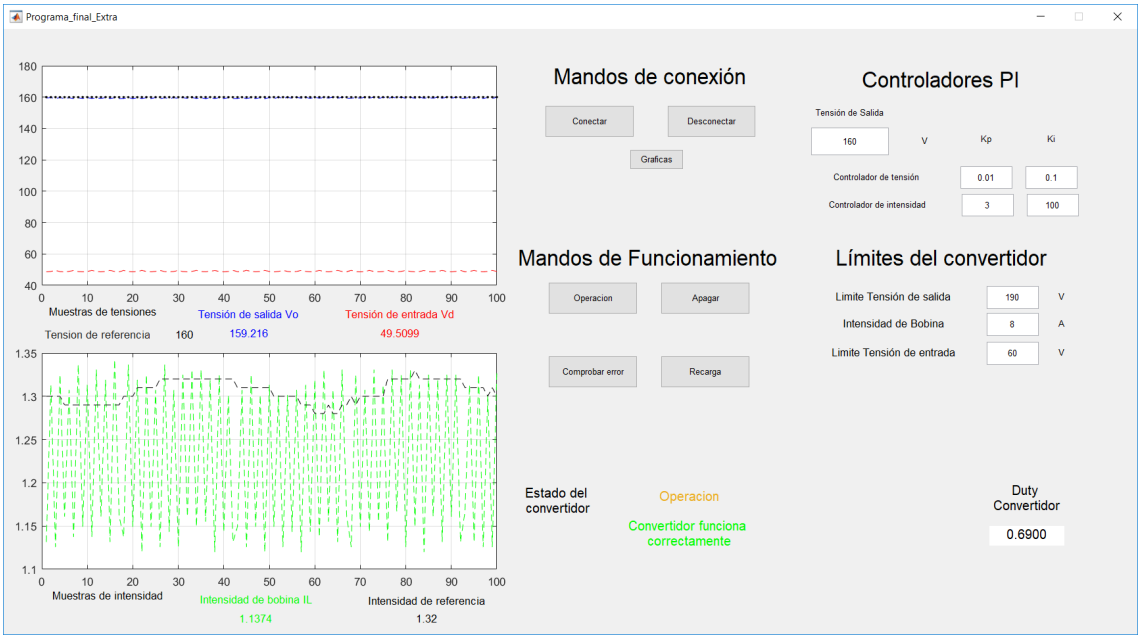


Figura 7-9. Tercera prueba. Medidas de la ventana de comandos.

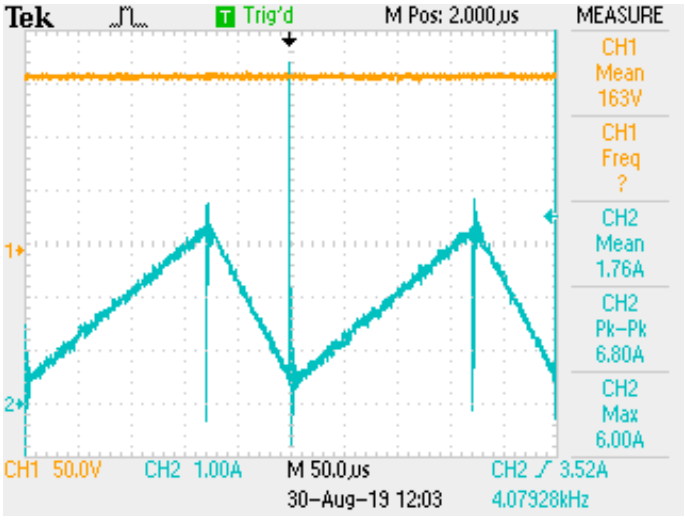


Figura 7-10. Tercera prueba. Medidas tomasdas por osciloscopio.

8 CONCLUSIONES

El proyecto ha cumplido sus objetivos:

- Se ha programado la DSP TMS320-F28377S mediante la herramienta simulink de Matlab, la cual se encarga de:
    - Tomar las medidas V_o, V_d e I_L del circuito a través de los módulos ADC.
    - Calcular con el diagrama de control construido el Duty Cycle necesario.
    - Implementar el duty calculado con la modulación del bloque PWM.
    - Implementar los distintos estados definidos.
    - Comunicarse con la interfaz diseñada, para transmisión y recepción de información a por el puerto serie.
- Análisis de los elementos electrónicos y eléctricos empleados en la construcción del elevador.
- Construcción del circuito elevador, comenzando por las conexiones entre sus elementos, así como el diseño de algunos de ellos, como los cables. Además, se ha instalado sujeto a una base de madera permitiendo el fácil transporte y puesta en marcha del circuito.
- Diseño de una interfaz gráfica usando la herramienta GUIDE de Matlab con la que se puede visualizar el funcionamiento del circuito, así como enviar las ordenes pertinentes al DSP, gracias a la comunicación en doble sentido (transmisión y recepción).
- Puesta en marcha del dispositivo, en conjunto de la interfaz y el circuito físico, comprobando su correcto funcionamiento mediante el uso de un osciloscopio.

Como posible ampliación del proyecto se podrían plantear los siguientes objetivos:

- Desarrollar una simulación consistente en la tensión salida de un parque fotovoltáico, la cual correspondería a la tensión de entrada del convertidor.
- Construir un dispositivo DC/AC, que transformase la tensión de salida del dispositivo elevador, de continua a alterna.

9 BIBLIOGRAFÍA

El datasheet de la placa TMS 320 F28377S se ha obtenido de la página de Texas Instrument:

http://www.ti.com/lit/ds/symlink/tms320f28377s.pdf

http://www.ti.com/tool/LAUNCHXL-F28377S

Se ha empleado la web que nos ofrece MATLAB de ayuda para para consultar las dudas del circuito, así como se ha hecho uso también del foro.

https://es.mathworks.com/help/matlab/

https://es.mathworks.com/help/index.html?s_tid=CRUX_lftnav

Se ha hecho uso a su vez de la página web youtube, para consultar de forma visual distintos problemas que han ido surgiendo en el desarrollo de este proyecto, como el uso de la herramienta GUIDE.

https://www.youtube.com/

En el departamento de electrónica se han suministrado los datasheets, y esquemáticos de los distintos circuitos empleados para construir el dispositivo elevador.

Además se ha buscado en distintas páginas web las características de los componentes que han sido necesarios.

Se adjunta una recopilación de ellos en una carpeta de Google drive:

https://drive.google.com/drive/folders/12r4WpFVkYpaEyc2uC2jUj23E8_tkhF_L?usp=sharing

Para la teoría a cerca del dispositivo elevador se han consultado los apuntes de teoría y problemas de la asignatura “Electrónica de Potencia” de tercer curso de GITI.

10 ANEXO. CÓDIGOS DE MATLAB

1. **Configurar puerto COMx.**

```
setpref('MathWorks_Embedded_IDE_Link_PIL_Preferences','COMPort','COM4');

setpref('MathWorks_Embedded_IDE_Link_PIL_Preferences','BaudRate',115200);

setpref('MathWorks_Embedded_IDE_Link_PIL_Preferences','enableserial', true);
```

2. **Guardo 100 medidas.**

```
function Guardo_100_Medidas(vo,il,vd)
global i Vo_bus IL_bus Vd_bus enable_flag

    if enable_flag==0
        %Si la bandera no está activa, copio las medidas de los ADC en los
        %buses de datos de las variables.
        Vo_bus(i,1)=vo;
        IL_bus(i,1)=il;
        Vd_bus(i,1)=vd;
        i=i+uint16(1);

        %Si lleno el buffer, activo la bandera que habilita el cambio de
estados
        %y reinicio el contador.
        if i==101
            i=uint16(1);
            enable_flag=uint16(100);
        end
    end
end
```

3. **Evaluo errores y transitorios.**

```
function Evaluo_Errores_y_Transitorios
global ERROR ERROR_PH Limite_Vo Limite_IL Limite_Vd
global MENSAJE Vo_l IL_l Vd_l TRANSITORIO Vo_sal


    %Evaluo si las medidas están dentro de los valores admisibles impuestos
    %Se ha redondeado a 5 decimas para transformar los datos a tipo double

    %Si no estamos en el estado error, evaluamos si hay algún error.
    if MENSAJE~=4
        if ERROR_PH>0
            ERROR=uint16(1);
        elseif Vo_l>single(Limite_Vo)
            ERROR=uint16(2);
        elseif IL_l>single(Limite_IL)
            ERROR=uint16(3);
        elseif Vd_l>single(Limite_Vd)
            ERROR=uint16(4);
        else
            ERROR=uint16(0);
        end
    end
```

```
    %Defino como objetivo para pasar los estados transitorios, que se
    %alcance la mitad de la tensión de salida deseada

    if MENSAJE==1
        if Vo_l<0.75*Vo_sal
            TRANSITORIO=uint16(0);
        else
            TRANSITORIO=uint16(2);
        end
    end
    if MENSAJE==3
        if Vo_l<Vd_l
            TRANSITORIO=uint16(0);
        else
            TRANSITORIO=uint16(4);
        end
    end
end
```

4. **Envio 100.**

```
function envio_100
global j Vo_bus IL_bus Vd_bus s_mandar enable_flag VoSCI ILSCI VdSCI enable

%Si recibo señal de mandar

if s_mandar==1 && enable==1 && j<101

    %Envio la información que tengo guardada.
    %No estoy enviando paquete de datos, si no más bien una corriente
    %en caso de que me estén pidiendo información
        VoSCI=Vo_bus(j,1);
        ILSCI=IL_bus(j,1);
        VdSCI=Vd_bus(j,1);
        j=j+uint16(1);
    if j==101
        j=uint16(1);
        enable_flag=uint16(0);
    end
end
end
```

5. **Mido ADC puerto Serie.**

```
%Defino las variables que quiero obtener
aVo=zeros(1,50);
bIL=zeros(1,50);
cVd=zeros(1,50);

%Elimino cualquier acceso al COM4
delete(instrfindall);

%Me conecto al puerto COM4
s=serial('COM4');
s.BaudRate=115200;
fopen(s);

%Inicializo variables
j=1;
```

```
i=1;
n_muestras=51;

    entrada_adc = fread(s,150,'uint16');

    Entrada=char(typecast(uint16(entrada_adc),'uint8'));

while i<n_muestras
    %Tomo 100 muestras


 if ((Entrada(j)=='A'))

          zVo_8bits=uint8([Entrada(j+1) Entrada(j+2)]);
             zIL_8bits=uint8([Entrada(j+3) Entrada(j+4)]);
             zVd_8bits=uint8([Entrada(j+5) Entrada(j+6)]);

             ok_proto_8bits=uint8([Entrada(j+7) Entrada(j+8)]);
             ok_proto=typecast(ok_proto_8bits,'uint16');


        if ok_proto==uint16(100)
             aVo(i)=typecast(zVo_8bits,'uint16');
             bIL(i)=typecast(zIL_8bits,'uint16');
             cVd(i)=typecast(zVd_8bits,'uint16');
             i=i+1;

        end

        j=j+1;
    elseif j>150
        j=1;
         entrada_adc = fread(s,50,'uint16');
    else
        j=j+1;
 end
end
Media_Vo=mean(aVo);
Media_IL=mean(bIL);
Media_Vd=mean(cVd);

 fclose(s);
 delete(s);
%clear all;
%clc;
```

6. Calculo rectas regresión

```
%Valores de tensión a medir
V=[0.3,5,10,25,50,75,100,125,150,175,200];
%Valores del ADC a las tensiones medidas
Vo=[ 21.06, 121.66, 225.1, 529.34, 1037.3, 1547.8, 2053.4, 2561.5, 3071.5,
3581.7, 4095];
Vd=[ 23.48, 118.04, 215.64, 498.54, 973.9, 1443.2, 1917.5, 2390.3, 2862.7,
3333.8, 3812.2];

%Valores de Intensidad a medir
I=[0,0.25, 0.5, 1, 1.5, 2, 3, 4, 5, 5.25];
%Valores del ADC a las intensidades medidas
IL=[1929.1, 1970.5, 2014.1, 2098.6, 2184.2, 2267.9, 2437.3,
2609.4,2780.4,2823.7];
```

```
%Cambio de formato para visualizar más cifras decimales
format long


%Para visualizar las rectas, las dibujo al mismo tiempo que calculo
%mediante la función polyfit las incógnitas de las rectas de regresión que
%necesito para mi diagrama.

plot(V,Vo)
pVo=(polyfit(Vo,V,1))

hold on

plot(V,Vd)
pVd=polyfit(Vd,V,1)

hold on
pIL=polyfit(IL,I,1)
```

7. **Seleccionar datos.**

```
function fcn(u)
global ESTADOS s_mandar Vo_sal PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki
global Limite_Vo Limite_IL Limite_Vd
ESTADOS=u(1);
s_mandar=u(2);
Vo_sal=single(u(3))/32;
PI_V_Kp=single(u(4))/100;
PI_V_Ki=single(u(5))/32;
PI_I_Kp=single(u(6))/32;
PI_I_Ki=single(u(7))/32;
Limite_Vo=single(u(8))/32;
Limite_IL=single(u(9))/32;
Limite_Vd=single(u(10))/32;
end
```

8. **Conversor Duty.**

```
function y = Conv_duty(u)
y =uint16(u*1000);
end
```

9. **Conversor Duty.**

```
function y = Conv_I_ref(u)
y =uint16(u*100);
end
```

10. Programa final Extra.

```
function varargout = Programa_final_Extra(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Programa_final_Extra_OpeningFcn, ...
                   'gui_OutputFcn',  @Programa_final_Extra_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
end
function Programa_final_Extra_OpeningFcn(hObject, eventdata, handles,
varargin)
% Choose default command line output for Programa_final_Extra
handles.output = hObject;
set(handles.estado,'string', sprintf('TFG'));
set(handles.tipoerr,'string', sprintf('Guillermo Hidalgo González'));
set(handles.error,'string', sprintf('GITI Electrónico'));
set(handles.correcto,'string', sprintf(' '));
pause(1);
set(handles.estado,'string', sprintf('Pulsar conectar'));
set(handles.correcto,'string', sprintf('Comprobar límites del
convertidor'));

% Update handles structure
guidata(hObject, handles);
end

function varargout = Programa_final_Extra_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;
end

% Cuando pulsemos el botón conecta
function conecta_Callback(hObject, eventdata, handles)
global s encendido
global t Vo IL Vd i j contador pedir_datos envio_micro estado I_ref
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki Limite_Vd Limite_IL Limite_Vo
handles.output = hObject;
guidata(hObject, handles);

%Definimos las variables globales y las inicializamos
Vo=zeros(1,100);
IL=zeros(1,100);
Vd=zeros(1,100);
I_ref=zeros(1,100);
i=1;
j=1;
contador=zeros(1,100);
estado=0;
pedir_datos=0;

%Definimos un reloj que cada 1 ms ejecute la función dibuja
```

```
t=timer('TimerFcn',{@dibuja,
handles},'Period',0.001,'ExecutionMode','fixedSpacing');

%Obtengo el valor de todas las variables que puedo definir en la interfaz
    Vo_des=str2num(get(handles.vo_ref,'string'))*32;
    PI_V_Kp=str2num(get(handles.PIvKp,'string'))*100;
    PI_V_Ki=str2num(get(handles.PIvKi,'string'))*32;
    PI_I_Kp=str2num(get(handles.PIiKp,'string'))*32;
    PI_I_Ki=str2num(get(handles.PIiKi,'string'))*32;
    Limite_Vo=str2num(get(handles.Limit_Vo,'string'))*32;
    Limite_IL=str2num(get(handles.Limit_IL,'string'))*32;
    Limite_Vd=str2num(get(handles.Limit_Vd,'string'))*32;

envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];


%Reseteamos las dos gráficas
axes(handles.grafica_tension);
cla reset;
grid(handles.grafica_tension, 'on');
axes(handles.grafica_intensidad);
cla reset;
grid(handles.grafica_intensidad, 'on');

%Nos conectamos al puerto COMx mediante comunicación en serie
delete(instrfindall);
s=serial('COM4','BaudRate',115200);
fopen(s);

%Enviamos la orden para inicializar el programa del microcontrolador
fwrite(s,envio_micro,'uint16');
encendido=1;

%Escribimos Conectado en la ventana de texto habilitada
set(handles.duty_cycle,'string', sprintf(' '));
set(handles.estado,'string', sprintf('Conectado'));
set(handles.error,'string', sprintf(' '));
set(handles.tipoerr,'string', sprintf(' '));
set(handles.correcto,'string', sprintf('Iniciar adquisición de medidas'));
end

%Cuando pulsemos el botón desconecta
function desconecta_Callback(hObject, eventdata, handles)
global s encendido t i j contador pedir_datos envio_micro
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki estado Limite_Vo Limite_IL
Limite_Vd
%Si la conexión con el puerto serie está operativa
if(encendido==1)
    pedir_datos=0;
    envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    fwrite(s,envio_micro,'uint16');

    axes(handles.grafica_tension);
    cla reset;
    grid(handles.grafica_tension, 'on');
    axes(handles.grafica_intensidad);
    cla reset;
    grid(handles.grafica_intensidad, 'on');

    %Cerramos la conexión
```

```
    fclose(s);
    delete(instrfindall);
    encendido=0;

    %Eliminamos el temporizador
    delete(t);

    %Inicializamos las variables globales
    j=1;
    i=1;
    contador=1;
end

%Escribimos Desconectado en la ventana de texto habilitada
set(handles.estado,'string', sprintf('Desconectado'));
set(handles.error,'string', sprintf(' '));
clc;
end

%Cuando pulsemos el botón operación
function operacion_Callback(hObject, eventdata, handles)
global s encendido estado envio_micro
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos Limite_Vo
Limite_IL Limite_Vd
if encendido==1
    %Enviamos la orden para pasar al estado de operacion
    estado=1;
    envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    fwrite(s,envio_micro,'uint16');
end
end


%Cuando pulsemos el botón apagar.
function apagar_Callback(hObject, eventdata, handles)
global s encendido estado envio_micro
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos Limite_Vo
Limite_IL Limite_Vd
if encendido==1
    %Enviamos la orden para pasar al estado de apagar
    estado=3;
    envio_micro=[estado,pedir_datos,Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    fwrite(s,envio_micro,'uint16');
end
end

%Cuando pulsemos el botón Comprobar error.
function Comprobar_error_Callback(hObject, eventdata, handles)
global s encendido  estado envio_micro
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos Limite_Vo
Limite_IL Limite_Vd
if encendido==1
    %Enviamos la orden para pasar al estado de recarga
    estado=uint16(7);
    envio_micro=[estado,pedir_datos,Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    fwrite(s,envio_micro,'uint16');
    set(handles.error,'string', sprintf('Comprobar Medidas'));
end
end

%Cuando pulsemos el botón recarga.
```

```
function recarga_Callback(hObject, eventdata, handles)
global s encendido  estado envio_micro
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos Limite_Vo
Limite_IL Limite_Vd
if encendido==1
    %Enviamos la orden para pasar al estado de recarga
    estado=uint16(8);
    envio_micro=[estado,pedir_datos,Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    fwrite(s,envio_micro,'uint16');
end
end

%Cuando pulsemos el botón grafica.
function grafica_Callback(hObject, eventdata, handles)
global  t s salida Entrada envio_micro estado
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos Limite_Vo
Limite_IL Limite_Vd

%Transitorio de 3 segundos
set(handles.estado,'string', sprintf('.'));
pause(0.5);
set(handles.estado,'string', sprintf('..'));
pause(0.5);
set(handles.estado,'string', sprintf('...'));
pause(0.5);

%Inicio el timer
    pedir_datos=1;
    envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    %Envio señal para iniciar envio de informacion
    fwrite(s,envio_micro,'uint16');

    %Empiezo a leer una vez he activado la transferencia
    salida = fread(s,100,'uint16');
    Entrada=char(typecast(uint16(salida),'uint8'));
    set(handles.estado,'string', sprintf('Tomando medidas'));
    start(t);
end

%Función que se ejecuta en cada periodo del reloj de 1 ms
function dibuja(hObject, eventdata, handles)
global s encendido Vo IL Vd i j contador salida estado  Entrada nueva_medida
global  error_conv Vo_des I_ref

if encendido==1
    %Protocolo de comunicación, donde los datos que llegan deban pasar
    %una serie de filtros, para asegurar que la información recibida es
    %veráz
    if j<150
        if ((Entrada(j)=='A')&&(Entrada(j+17)=='B'))
            nueva_medida=0;
            zVo_8bits=uint8([Entrada(j+1) Entrada(j+2)]);
            zIL_8bits=uint8([Entrada(j+3) Entrada(j+4)]);
            zVd_8bits=uint8([Entrada(j+5) Entrada(j+6)]);

            ok_proto_8bits=uint8([Entrada(j+7) Entrada(j+8)]);
            ok_proto=typecast(ok_proto_8bits,'uint16');

            duty_8bits=uint8([Entrada(j+9) Entrada(j+10)]);
            duty=typecast(duty_8bits,'uint16');
```

```
            duty_cycle=double(duty)*0.001;

            estado_8bits=uint8([Entrada(j+11) Entrada(j+12)]);
            estado=typecast(estado_8bits,'uint16');

            error_8bits=uint8([Entrada(j+13) Entrada(j+14)]);
            error_conv=typecast(error_8bits,'uint16');

            I_ref_8bits=uint8([Entrada(j+15) Entrada(j+16)]);
            I_ref_conv=typecast(I_ref_8bits,'uint16');
            I_ref(i)=single(I_ref_conv)*0.01;
            IL_error=I_ref(i);

            if ok_proto==uint16(100)

                %Puesto que el envío por la comunicación SCI es más
eficiente
                %en un numero entero, realizamos la conversión aqui del
valor
                %ADC al de Tensión e Intensidad
                Voo=typecast(zVo_8bits,'uint16');
                ILL=typecast(zIL_8bits,'uint16');
                Vdd=typecast(zVd_8bits,'uint16');
                Vo(i)=double(Voo)*0.049076476117753-0.529622501407953;
                Vocifra=Vo(i);
                IL(i)=double(ILL)*0.005870195030470-11.319073216832791;
                ILcifra=IL(i);
                Vd(i)=double(Vdd)*0.052207616214077+0.149116742056191;
                Vdcifra=Vd(i);
                contador(i)=i;
                i=i+1;
                Duty=sprintf('%.4f',duty_cycle);
                set(handles.duty_cycle,'string', Duty);
                set(handles.Vo_cifra,'string', Vocifra);
                set(handles.Vd_cifra,'string', Vdcifra);
                set(handles.IL_cifra,'string', ILcifra);
                set(handles.IL_error,'string', IL_error);
                set(handles.v_reff,'string', Vo_des/32);

         %Compruebo el estado en el que se encuentra operando el
         %convertidor, y si se está produciendo algún tipo de error
         switch estado
            case 0
                    set(handles.estado,'string', sprintf('Apagado'));
            case 1
                    set(handles.estado,'string', sprintf('Precarga'));
            case 2
                    set(handles.estado,'string', sprintf('Operacion'));
            case 3
                    set(handles.estado,'string', sprintf('Descarga'));
            case 4
                    set(handles.estado,'string', sprintf('Error en el
Convertidor'));
                    set(handles.correcto,'string', sprintf('Ajustar'));
                    set(handles.tipoerr,'string', sprintf('Tipo de error'));
         end

         switch error_conv
            case 0
                if estado ~=4
                    set(handles.correcto,'string', sprintf('Convertidor
funciona correctamente'));
                    set(handles.error,'string', sprintf(' '));
                    set(handles.tipoerr,'string', sprintf(' '));
```

```
                end
            case 1
                set(handles.error,'string', sprintf('Error en Puente en
H'));
            case 2
                set(handles.error,'string', sprintf('Límite de Vo
alcanzado'));
            case 3
                set(handles.error,'string', sprintf('Límite de IL
alcanzado'));
            case 4
                set(handles.error,'string', sprintf('Límite de Vd
alcanzado'));
         end
            end
            j=j+12;

            %Cuando tengamos 100 medidas tomadas las representamos en las
            %gráficas habilitadas
            if i==101
                Vo_ref=Vo_des/32;
                Vo_limite=Vo_ref+10;
                axes(handles.grafica_tension);
                plot(handles.grafica_tension,contador,Vo,'b--
',contador,Vd,'r--',contador,Vo_limite,'w', contador,Vo_ref,'k.');
                grid(handles.grafica_tension, 'on');
                axes(handles.grafica_intensidad);
                plot(handles.grafica_intensidad,contador,IL,'g--
',contador,I_ref,'k--');
                grid(handles.grafica_intensidad, 'on');
                i=1;
            end
        else
            j=j+1;
        end
    elseif j>=150
        j=1;
        salida = fread(s,150,'uint16');
        Entrada=char(typecast(uint16(salida),'uint8'));
    end
end
end


%Funciones que recogen los valores de los parámetros que podemos definir en
%la intefaz
function vo_ref_Callback(hObject, eventdata, handles)
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos envio_micro estado
s
global Limite_Vo Limite_IL Limite_Vd
    Vo_des=str2num(get(handles.vo_ref,'string'))*32;
    envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    fwrite(s,envio_micro,'uint16');
end
function vo_ref_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end
```

```
function PIvKp_Callback(hObject, eventdata, handles)
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos envio_micro estado
s
global Limite_Vo Limite_IL Limite_Vd
    PI_V_Kp=str2num(get(handles.PIvKp,'string'))*100;
    envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    fwrite(s,envio_micro,'uint16');
end
function PIvKp_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

function PIiKp_Callback(hObject, eventdata, handles)
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos envio_micro estado
s
global Limite_Vo Limite_IL Limite_Vd
    PI_I_Kp=str2num(get(handles.PIiKp,'string'))*32;
    envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    fwrite(s,envio_micro,'uint16');
end
function PIiKp_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

function PIvKi_Callback(hObject, eventdata, handles)
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos envio_micro estado
s
global Limite_Vo Limite_IL Limite_Vd
    PI_V_Ki=str2num(get(handles.PIvKi,'string'))*32;
    envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    fwrite(s,envio_micro,'uint16');
end
function PIvKi_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

function PIiKi_Callback(hObject, eventdata, handles)
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos envio_micro estado
s
global Limite_Vo Limite_IL Limite_Vd
  PI_I_Ki=str2num(get(handles.PIiKi,'string'))*32;
  envio_micro=[estado,pedir_datos,  Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
  fwrite(s,envio_micro,'uint16');
end
function PIiKi_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end
```

```
function Limit_Vo_Callback(hObject, eventdata, handles)
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos envio_micro estado
s
global Limite_Vo Limite_IL Limite_Vd
    Limite_Vo=str2num(get(handles.Limit_Vo,'string'))*32;
     envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
     fwrite(s,envio_micro,'uint16');
end
function Limit_Vo_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

function Limit_IL_Callback(hObject, eventdata, handles)
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos envio_micro estado
s
global Limite_Vo Limite_IL Limite_Vd
    Limite_IL=str2num(get(handles.Limit_IL,'string'))*32;
     envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
     fwrite(s,envio_micro,'uint16');
end
function Limit_IL_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

function Limit_Vd_Callback(hObject, eventdata, handles)
global Vo_des PI_V_Kp PI_V_Ki PI_I_Kp PI_I_Ki pedir_datos envio_micro estado
s
global Limite_Vo Limite_IL Limite_Vd
    Limite_Vd=str2num(get(handles.Limit_Vd,'string'))*32;
    envio_micro=[estado,pedir_datos, Vo_des, PI_V_Kp, PI_V_Ki,
PI_I_Kp,PI_I_Ki,Limite_Vo,Limite_IL,Limite_Vd];
    fwrite(s,envio_micro,'uint16');
end
function Limit_Vd_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end
```